(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
24 September 2020 (24.09.2020)

WIPO | PCT

(10) International Publication Number
WO 2020/190736 A1

(51) International Patent Classification:
A63F 13/56 (2014.01) *G06N 20/00* (2019.01)
G06F 3/01 (2006.01) *G06T 19/20* (2011.01)

(21) International Application Number:
PCT/US2020/022670

(22) International Filing Date:
13 March 2020 (13.03.2020)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
62/819,193 15 March 2019 (15.03.2019) US

(71) Applicant: **RCT STUDIO INC.** [US/US]; 1003 Chestnut Street, Burbank, California 91596 (US).

(72) Inventors: **LYU, Cheng**; c/o RCT Studio Inc., 1003 Chestnut Street, Burbank, California 91596 (US). **CHEN, Yuheng**; c/o RCT Studio Inc., 1003 Chestnut Street, Burbank, California 91596 (US). **LI, Xiang**; c/o RCT Studio Inc., 1003 Chestnut Street, Burbank, California 91596 (US). **MOU, Xiaolong**; c/o RCT Studio Inc., 1003 Chestnut Street, Burbank, California 91596 (US). **WANG, Shoukun**; c/o RCT Studio Inc., 1003 Chestnut Street, Burbank, California 91596 (US). **HONG, Qiangning**; c/o RCT Studio Inc., 1003 Chestnut Street, Burbank, California 91596 (US). **ZHANG, Yan**; c/o RCT Studio Inc., 1003 Chestnut Street, Burbank, California 91596 (US).

(74) Agent: **FOWLER, Colin** et al.; Perkins Coie LLP, P.O. Box 1247, Seattle, WA 98111 -1247 (US).

(81) Designated States *(unless otherwise indicated, for every kind of national protection available)*: AE, AG, AL, AM, AO, AT, AU, AZ, BA, BB, BG, BH, BN, BR, BW, BY, BZ, CA, CH, CL, CN, CO, CR, CU, CZ, DE, DJ, DK, DM, DO, DZ, EC, EE, EG, ES, FI, GB, GD, GE, GH, GM, GT, HN, HR, HU, ID, IL, IN, IR, IS, JO, JP, KE, KG, KH, KN, KP, KR, KW, KZ, LA, LC, LK, LR, LS, LU, LY, MA, MD, ME, MG, MK, MN, MW, MX, MY, MZ, NA, NG, NI, NO, NZ, OM, PA, PE, PG, PH, PL, PT, QA, RO, RS, RU, RW, SA, SC, SD, SE, SG, SK, SL, ST, SV, SY, TH, TJ, TM, TN, TR, TT, TZ, UA, UG, US, UZ, VC, VN, WS, ZA, ZM, ZW.

(84) Designated States *(unless otherwise indicated, for every kind of regional protection available)*: ARIPO (BW, GH, GM, KE, LR, LS, MW, MZ, NA, RW, SD, SL, ST, SZ, TZ, UG, ZM, ZW), Eurasian (AM, AZ, BY, KG, KZ, RU, TJ, TM), European (AL, AT, BE, BG, CH, CY, CZ, DE, DK, EE, ES, FI, FR, GB, GR, HR, HU, IE, IS, IT, LT, LU, LV, MC, MK, MT, NL, NO, PL, PT, RO, RS, SE, SI, SK, SM, TR), OAPI (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, KM, ML, MR, NE, SN, TD, TG).

Declarations under Rule 4.17:
— *as to applicant's entitlement to apply for and be granted a patent (Rule 4.17(ii))*
— *as to the applicant's entitlement to claim the priority of the earlier application (Rule 4.17(iii))*

(54) **Title:** METHODS, SYSTEMS, AND APPARATUSES FOR PRODUCTION OF AN INTERACTIVE MOVIE

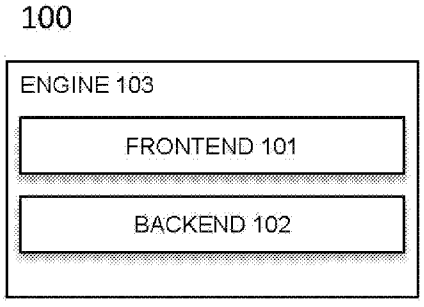


FIG. 1

(57) **Abstract:** Various examples are directed to systems and methods for producing an interactive movie. Such a method may include analyzing a script to generate a scene, a character, or a story; generating a chaos box for the scene, character, or story, the chaos box including a plurality of basic units each having at least one variable of the scene, character, or story; assigning a first value to the at least one variable of the scene, character, or story in each basic unit of the chaos box; comparing a second value of the at least one variable of an input from a user with the first value of the basic unit in the chaos box; and selecting an exit from the chaos box based on the comparison.

WO 2020/190736 A1

[Continued on next page]

Published:
— *with international search report (Art. 21(3))*

METHODS, SYSTEMS, AND APPARATUSES FOR PRODUCTION OF AN INTERACTIVE MOVIE

CROSS REFERENCE TO RELATED APPLICATION

The present application claims priority to U.S. Provisional Application No. 62/819,193, filed on March 15, 2019, the entire contents of which are being incorporated herein by reference.

BACKGROUND

The present disclosure relates generally to the field of game and film production, particularly, production of interactive games and/or interactive movies. In a traditional movie, there is usually only one linear storyline, and the storyline cannot be created or influenced by the viewer. In a computer game, the storyline may be influenced by the player if choice points, i.e., story branches, have been hard-coded into the game. However, the number of choice points is usually small.

SUMMARY

According to an aspect of the present disclosure, a method comprise uploading a script to an engine by a script uploading unit of a computing system, the computing device comprising at least one processor and a data storage device in communication with the at least one processor; analyzing the script to generate at least one of a scene, a character, or a story by a script analyzing unit of the computing system; generating a chaos box for the at least one of a scene, a character, or a story, the chaos box comprising a plurality of basic units each comprising at least one variable of the at least one of a scene, a character, or a story; assigning a first value to the at least one variable of the at least one of a scene, a character, or a story in each basic unit of the chaos box; receiving an input comprising the at least one variable having a second value from a user by an input device; comparing the second value of the at least one variable of the input from the user with the first value of the at least one variable of the at least one of a scene, a character, or a story of the basic units in the chaos box; and selecting an exit from the chaos box based on the comparison.

According to a further aspect of the present disclosure, the method comprises adding to a basic unit a filter selected from the group consisting of an event type filter, an event context filter, an observer self-status filter, a related non-player character (NPC) status filter, a relationship filter, and combinations thereof.

According to a further aspect of the present disclosure, the script analyzing unit comprises at least one of a character extraction unit, a scene parser, an entity extraction, or a modifier phrase parser unit.

According to a further aspect of the present disclosure, the input from the user is selected from the group consisting of a non-voice input, a voice input, and a combination thereof. The non-voice input may be collected by a controller and/or sensor, which may be part of or connected and

compatible with the computing system. The input voice may be collected by a microphone, which may be part of the computing system or part of a device other than the computing system.

According to a further aspect of the present disclosure, the input from the user is a voice input, the method comprising converting the voice input to texts by an audio-to-text (ASR) unit of the computing device.

According to a further aspect of the present disclosure, the input from the user is a non-voice input, the method comprising receiving and analyzing the non-voice input by a controller and/or a sensor.

According to a further aspect of the present disclosure, the method further comprises editing the at least one of a scene, a character, a story, or a basic unit by changing the first value of the at least one variable of the at least one of a scene, a character, a story, or a basic unit.

According to a further aspect of the present disclosure, the method further comprises generating a new basic unit based on a pattern of the editing of the at least one of a scene, a character, or a story learned by the computing device.

According to a further aspect of the present disclosure, a system comprises at least one processor; a data storage device in communication with the at least one processor, wherein the data storage device comprises instructions that, when executed by the at least one processor, cause the at least one processor to analyze a script to generate at least one of a scene, a character, or a story; generate a chaos box for the at least one of a scene, a character, or a story, the chaos box comprising a plurality of basic units each comprising at least one variable of the at least one of a scene, a character, or a story; assign a first value to the at least one variable of the at least one of a scene, a character, or a story in each basic unit of the chaos box; compare a second value of the at least one variable of an input from a user with the first value of the at least one variable of the at least one of a scene, a character, or a story of the basic units in the chaos box; and select an exit from the chaos box based on the comparison.

According to a further aspect of the present disclosure, a non-transitory computer-readable medium storing software comprising instructions executable by at least one processor which, upon such execution, cause the at least one processor to perform operations comprising: analyzing a script to generate at least one of a scene, a character, or a story; generating a chaos box for the at least one of a scene, a character, or a story, the chaos box comprising a plurality of basic units each comprising at least one variable of the at least one of a scene, a character, or a story; assigning a first value to the at least one variable of the at least one of a scene, a character, or a story in each basic unit of the chaos box; comparing a second value of the at least one variable of an input from a user with the first value of the at least one variable of the at least one of a scene, a character, or a story of the basic units in the chaos box; and selecting an exit from the chaos box based on the comparison.

It may be appreciated by one of skill in the art that one or more various aspects of the disclosure may include but are not limited to circuitry and/or programming for effecting the herein-referenced aspects of the present disclosure; the circuitry and/or programming may be virtually

any combination of hardware, software, and/or firmware configured to effect the herein-referenced aspects depending upon the design choices of the system designer.

Further, the foregoing is a summary and thus contains, by necessity, simplifications, generalizations and omissions of detail. Those skilled in the art will appreciate that the summary is illustrative only and is not intended to be limiting in any way.

BRIEF DESCRIPTION OF THE DRAWINGS

Various examples are described herein in conjunction with the following figures, wherein:

FIG. 1 is a high-level component diagram of an example system according to an example embodiment of the present disclosure.

FIG. 2 shows an example embodiment of the present disclosure.

FIG. 3 shows an example frontend 300 according to an example embodiment of the present disclosure.

FIG. 4 shows an example backend 400 according to an example embodiment of the present disclosure.

FIG. 5 is a flow chart showing one example of scene editing according to an example embodiment of the present disclosure.

FIG. 6 is a flow chart showing one example of character editing according to an example embodiment of the present disclosure.

FIG. 7 is a flow chart showing one example of story editing according to an example embodiment of the present disclosure.

FIG. 8 is a flow chart showing one example of chaos box editing according to an example embodiment of the present disclosure.

FIG. 9 is a flow chart showing one example of action and rendering according to an example embodiment of the present disclosure.

FIG. 10 shows an example script analyzing and/or RRNLS unit 1000 according to an example embodiment of the present disclosure.

FIG. 11 shows an example interactive movie progression according to the present disclosure.

FIG. 12 shows an example basic unit 1200 of a chaos box according to an example embodiment of the present disclosure.

FIG. 13 shows an example basic unit 1300 of a chaos box according to an example embodiment of the present disclosure.

FIG. 14 shows an example system 1400 according to an example embodiment of the present disclosure.

FIG. 15 is a high-level component diagram of an example computing device 1500 performing the present disclosure or any portion thereof according to an example embodiment of the present disclosure.

FIG. 16 is a component diagram of an example computing device 1600 operating as any of nodes 1510A-B and/or any virtual machine in FIG. 15 according to an example embodiment of the present disclosure.

FIG. 17 shows an example network in which the computing device 1600 may operate according to an example embodiment of the present disclosure.

FIG. 18 shows an example of chaos boxes according to an example embodiment of the present disclosure.

FIG. 19 shows an example procedure in a Basic Unit of a chaos box according to an example embodiment of the present disclosure.

FIGS. 20A-20Z are example user interfaces according to n example embodiment of the present disclosure performing the steps in FIGS. 5-9.

FIG. 21 shows an example RRNLS (Real-time rendering from natural language script) NLU process flow chart.

FIG. 22 shows an example simulation and training process.

FIG. 23 shows an example process of interaction between an agent and the environment.

FIG. 24 shows an example structure diagram of the integrated training.

FIGS. 25A and 25B show respective example cumulative income curves of Chance and Besty during a training session (30,000 episodes).

FIG. 26A shows an example cumulative income curve of Besty during a training session (30,000 episodes).

FIG. 26B shows an example cumulative income curve of Chance during the same training session (30,000 episodes).

FIG. 27 shows rewards an example curve of an agent.

FIG. 28 shows an example of visualization of the predicted effects of the model at different stages.

FIG. 29 shows an example flow chart of the present invention.

DETAILED DESCRIPTION OF EXAMPLE EMBODIMENTS

Various examples are directed to methods, systems, and apparatus, including computer programs encoded on a computer storage medium, for creating an interactive movie that may provide the viewer with an open experience.

In various examples, the present disclosure may provide highly immersive interactive movies using Artificial Intelligence including, for example, limited-field natural language processing (NLP) and machine learning, for example, in the script generating process. In such an interactive movie, a viewer may assume the role of a character in a fictional virtual world, for example, making their own decisions and taking their own actions, to push the story forward. The viewer may or may not select a decision, action, or line from a given list of decisions, actions, or lines. The viewer may have one of a kind experience every time they view the movie.

In various examples, the present disclosure may be used in the script creation process for interactive games and/or movies. The present disclosure may greatly improve the efficiency of

scriptwriters and/or directors. The present disclosure may generate a lot of new storylines, scenes, and/or scene progressions for scriptwriters and/or directors. The present disclosure may allow the director to quickly preview the possibilities that may be generated by each behavior and/or action of a character in a virtual world, including the outcome or reactions from the environment and other characters.

Used in interactive movies, the present disclosure may greatly enhance the audience's experience. The present disclosure is distinguished from the operation mode of the traditional interactive movies that have only limited fixed story branches. With the present disclosure, the audience, including but not limited to movie goers, movie viewers, directors, screenwriters, game players, and game viewers, may have a different experience every time they watch the movie, and these experiences may be non-replicable.

In various examples, the present disclosure may or may not use events as the basic unit to define the possibility of a story that will occur in a confined space. Using events as the basis unit may generate limited possibilities and require a lot of manpower. The present disclosure may automatically generate almost all possibilities by simulating the rules of operation in the real world using predetermined rules of operation in the virtual world, including physics rules and characters' logic rules.

In various examples, the present disclosure may include at least one chaos box. The chaos box may include basic units defining the elements of a portion of the movie, including, for example, the scene, the status of characters, and the entrance (beginning) of the status of the relationship between the characters and multiple exits (outcomes). The progression from the entrance to each exit of the chaos box may be based on the basic units of the character in the scene. At the entrance of the chaos box, the possibilities in the chaos box are chaotic, and it may be unknown which outcome the current storyline will have from this chaos box. The chaos box may include a large number of basic units. The rules of operation for each character in this chaos box may have been defined, and as the character continues to take actions and/or generate behaviors, based on the operation rules, the possibilities within the chaos box may continue to collapse and then reach one of the predetermined exits. The progression from the entrance to the exit may be uncertain and unpredictable at the time of entry, which may create more possibilities than traditional mechanisms that operate according to a fixed sequence of events. Therefore, the present disclosure may provide an open, infinite or at least almost infinite, experience for the user.

In various examples, the present disclosure may provide a basic unit of the chaos box with filters at various levels and quantify the variables in the basic unit. The variables may be assigned certain values, which may define the elements of at least a portion of the movie. Each chaos box may include a plurality of basic units, which may be a large number of basic units. Each basic unit may include a set of values for the variables, including key variables. Some example variables, including some example key variables, are described in detail later in this disclosure. The basic units may be independent of each other. The basic unit may be found, matched, or selected by matching the variables' values in the basic unit with the input values of the variables or the values of the variables at the "entrance" of the chaos box.

In various examples, the chaos box may allow human (e.g., thinking) logics and physics logics to be taken into account. The chaos box may be a real-world simulator. Any event input in the virtual world may converge through the chaos box to a fixed ending.

In various examples, the present disclosure may produce interactive films having non-linear storylines, and the viewer may experience changes in the storyline that may have been triggered by their own choice, action, behavior, emotion, feeling, status, dialogue, and/or response.

Reference will now be made in detail to various examples, several of which are illustrated in the accompanying figures. Wherever practical, similar or like reference numbers may be used in the figures and may indicate similar or like functionality. The figures depict examples of the disclosed systems or methods or apparatus for purposes of illustration only. One skilled in the art will readily recognize from the following description that alternative examples of the structures and methods illustrated herein may be employed without departing from the principles described herein.

In various examples, as shown in FIG. 1, the present disclosure may include a system 100 that includes a frontend 101 and a backend 102, both supported by an engine 103. The frontend 101 may be the movie client for the audience to experience the new interactive movies. The frontend 101 may run on a Virtual Reality (VR) headset. The backend 102 may be the editor client for movie makers. The movie makers may include directors, screenwriters, and animators, etc. The frontend 101 and the backend 102 may be both supported by the engine 103, and the engine 103 may maintain all the services for the frontend 101 and the backend 102.

In various examples, a user or viewer using, viewing, playing, and/or producing the interactive movie according to the present disclosure may be described in FIG. 2. The system according to the present disclosure may quantify the basic units in the chaos boxes and generate any number of or almost infinite possibilities according to rules that may be predetermined and/or automatically generated during the using, viewing, playing, and/or producing of the interactive movie.

As shown in FIG. 2, the observer's logic processor 201 may store and/or process logic scripts, such as npc_status, npc_relationship, value_effection, etc. The observer's logic processor 201 may be on a cloud brain 202. The cloud brain 202 may process event 204 and/or generate actions 205 with logic processors of characters, such as Non-Player Characters (NPCs). There may be a client physic engine 203. When a character says or does something, the physic engine 203 may process the action 205 and/or generate the event 204. The event 204 generated by the physic engine 203 may include hurt, sound, and/or movement, etc. The generated event 204 may be transmitted to the observer's logic processor 201. The cloud brain 202 may generate an action 205, such as use, speak, and/or move, etc. The generated action 205 may be transmitted to the client physic engine 203. The session 206 may be synchronized between the client and the cloud. The session 206 may include all the environmental data, such as entity status and relationship data, etc. Some example data structures according to the present disclosure will be described in detail later in this disclosure.

In the present disclosure, the frontend may include a physic engine, which may be within the client's graphic engine, a high quality shader, an event generation system, an observer system for characters, such as NPC, an audio-to-text converter or service (ASR), a real-world mapping rule, and/or an animation unit or a plurality of animation units from motion capture and/or rigging. As a non-limiting example, the frontend may use HTC VIVE Pro™ headset or kit as the environment. As another non-limiting example, the frontend may use Unity™ as the graphic engine.

FIG. 3 shows an example frontend 300 according to some aspects of the present disclosure. The frontend 300 may include an observation system 301. The observation system 301 may observe an event 302. The frontend 300 may also include an event system 303. The event system 303 may generate an event 302. The observation system 301 and the event system 303 may communicate with each other regarding the event 302. The frontend 300 may include a physic engine 304. The frontend 300 may also include a user device 308. The user device 308 may receive a non-voice input 307 from the user and transmit the received non-voice input 307 to the physic engine 304. The physic engine 304 may also receive and/or generate a non-voice input 307 and transmit the non-voice input 307 to the user device 308. The frontend 300 may include an ASR 305. The ASR 305 may receive a voice input 306 from the user device 308 and convert the received voice input 306 to texts. The ASR 305 may also transmit texts generated from an audio to the user device 308. The frontend 300 may include an NPC manager 309. The NPC manager 309 may send and/or receive actions 311 of the NPC to and/or from an engine 301. The engine 301 may be a server's engine service. The NPC manager 309 may push the story forward to the next decision point by prompting a new user input in the user device 308. The NPC manager 309 may generate some NPC actions until the user may interact with at least one NPC action generated by the NPC manager and have new input, which may lead to the next decision point. The observation system 301 may also communicate with the engine 310.

As shown in FIG. 4, the present disclosure may include at least one of the following backend modules: script initializing module 401, script writing module 402, script supervised self-evolve module 403, digital asset and physical module 404, and test and build module 405. As a non-limiting example, the backend may use iPad Pro™ as the environment.

The script initializing module 401 may be used by directors and/or screenwriters among others. A script initializing unit may be configured to analyze scripts, set up characters, and/or set up scenes and any other related items. The script initializing unit may import existing scripts and generate story block trees and basic units in the chaos boxes based on the imported scripts. The script initializing unit then may generate a list of characters, including the viewer and/or at least one NPC, as well as their actions, which may be modified later.

For example, as shown in FIG. 5, existing scripts may be uploaded 501. Then, one of the uploaded scripts may be selected 502 and analyzed 503. A list of scenes from the selected script may be generated 504. Each scene may be edited 505. Additionally or alternatively, as shown in FIG. 6, a list of characters based on the selected script may be generated 604. New characters may be added 605. Each character may be set up 606.

The script writing module 402 may be used by screenwriters among others. A script writing unit may add new basic units for the characters in different chaos boxes that may generate different stories. The script writing unit may also modify the content of existing basic units, for example, by changing the variables and/or changing the values of one or more variables and/or by editing the actions or behaviors from inputting natural language script.

The script supervised self-evolve module 403 may be used by designers for numerical setup, scripting, and/or tuning. The designers may be any designer involved in game and movie design/production, including but not limited to numerical designers, game player designers, and/or level designers. A script supervised self-evolve unit may be configured for numerical setup and/or tuning, basic unit tuning, and/or language script setup and/or tuning.

The designers may assign values to the variables for different basic units and/or make some tuning, for example, if the values in the basic units do not seem to be logical. When the designer is tuning the basic units, the computing device or system executing or implementing the present disclosure may learn the pattern of basic units' setup, for example, how to decide whether a basic unit is logical, so that the computing device or system may generate more possibilities, namely, new basic units that are different from the initial basic units. When the designer is tuning the basic units, a preview window may show the chain reaction of any modification that the designer has made and/or is making. The designer may supervise this machine learning process.

The digital asset and physical module 404 may be used by digital artists, cameramen, and/or visual effects artists among others. A digital asset and physical unit may be configured for importing facial expression, motion setup, camera and footage setup, and/or special effects setup. The digital asset and physical unit may import the digital asset for each NPC, items, and/or animations for all actions. The digital asset and physical unit may then tune the parameters of animations and expressions of the characters separately in different basic units, for example, the walking speed and/or the punching strength. Different models facial expression may be imported to match different emotions of the characters. The file formats of the models and the animations may be existing formats.

As shown in FIGS. 7 and 8, following script uploading 701, scene editing, 702 and/or character editing 703 may be story editing 704 using the above described modules, such as the script writing module, the script supervised self-evolve module, and/or the digital asset and physical module. A single 705 or a plurality 706 of stories may be selected. New branches may be added 707, where characters 715 and/or scenes 716 may be selected and/or newly added. Details 717 and summary 718 of the new branches may be added. Linkages may be determined between the scenes and/or characters and/or story progressions 708. Stories may be edited 709, copied 710, pasted 711, and/or deleted 712. Story editing 709, e.g., via editing the "card" that includes the content of the story 719, may include editing one or more of character(s) 720, scene(s) 721, summary 722, and/or detail(s) 723, each of which may be edited or newly added 724-731. Story tree(s) 713 may be generated. Based on the story tree(s), chaos boxes may be generated and/or edited 732. The viewer may preview to-be-generated chaos boxes 734 and may generate chaos boxes 735. A single 736 or a plurality 737 of chaos boxes may be selected. New branches within

a chaos box or among the chaos boxes may be added 738. Linkage(s) among the chaos boxes may be determined 739. The chaos box(es) may be edited 740, copied 741, pasted 742, and/or deleted 743. The chaos box editing may include action/behavior editing 745, which may include addition 746 and/or editing 747 of action(s)/behavior(s). The action/behavior editing may be in the format of tests editing 748 or space filling 749 and may involve change and/or selection of parameters 750.

As shown in FIG. 9, following script uploading 901, scene editing 902, character editing 903, and/or story editing 904 may be action and rendering 905. Story tree(s) 906 may be generated, and the details of the story or stories may be added 907. Chaos boxes may then be generated based on the story tree(s) 908. The generated chaos boxes may then be edited 909, including editing one or more of actions 910, scenes 911, and/or renderings 912.

In the test and build module 405, a test and build unit may be configured to allow the user to preview and watch each story block, iterate to optimize all the scripts, and/or output a playable and/or viewable version. The title may be determined, an introduction may be produced, and/or a platform for release may be selected.

In various examples, the engine may be configured for script analyzing and/or real-time rendering from Natural Language Script (RRNLS). FIG. 21 shows an example RRNLS process flow chart. The engine may include at least one chaos box. The engine may also include NLP module for frontend user input and/or logic engine for RRNLS. The engine may be configured for supervised machine learning (ML) in generating story possibilities. As a non-limiting example, the engine may use Node and/or Python as the environment. As another non-limiting example, the engine may use fastText and/or TensorFlow as the framework.

FIG. 10 shows an example script analyzing and/or RRNLS unit 1000 according to some aspects of the present disclosure. The script analyzing unit and the RRNLS unit may be the same unit or different units each including the same modules. The script analyzing and RRNLS unit 1000 may include a script uploading unit 1001, a scene parser 1002, a character extraction unit 1003, a character modifier extraction unit 1004, an entity extraction unit 1005, and an action data extraction unit 1006. The action data extraction unit 1006 may include a modifier phrase extraction unit 1007, an object extraction unit 1008, a target extraction unit 1009, a subject extraction unit 1010, and/or an action extraction unit 1011.

A script may be uploaded by the script uploading unit 1001. The scene parser 1002 may extract and/or generate scenes based on the uploaded script. The character extraction unit 1003 may extract and/or generate characters from the script. The character modifier extraction unit 1004 may modify the extracted and/or generated characters. The entity extraction unit 1005 may extract and/or generate the entities from the script. The action data extraction unit 1006 may extract and/or generate action data, such as object, target, subject, and action. The action extraction unit 1006 may extract and/or generate the action. The subject extraction unit 1010 may extract and/or generate the subject of the action. The object extraction unit 1008 may extract and/or generate the object of the action. The target extraction unit 1009 may extract and/or generate the target of the

action. The modifier phrase extraction unit 1007 may modify any of the extracted and/or generated action data.

In various examples, the chaos boxes may turn the low granularity of the story tree into a chaos status while still keeping the upper-level controlled and limited and therefore, may provide the user more open experience. As shown in FIG. 11, the present disclosure may provide non-linear branches in an interactive movie. After the story starts 1101, the movie may enter a chaos box 1102. The chaos box 1102 may include a large number of basic units and may include thousands of possibilities. Through a different exit, the movie may enter another chaos box 1103,1104, which may again provide thousands of possibilities. The movie may have a limited number of outcomes in the end 1105,1106, but the experiences of the audience from the start of the story to the ending may be almost infinite.

In various examples, as shown in FIG. 12, a basic unit 1200 of the chaos box may include filters, such as an event type filter, an event context filter, an observer self-status filter, a related NPC status filter, and/or a relationship filter. There may also be a reaction action list.

In various examples, as shown in FIG. 13, a basic unit 1300 of the chaos box may receive an event input 1301. The event input 1301 may be "what happened." The basic unit 1300 may include a script 1302. The script 1302 may be "what do 'I' suppose to react in this situation and event." After the script 1302 has been processed, i.e., the viewer/player has made their decision and taken an action. The action may result in new events. Other characters, such as other NPC may observe these new events and react to them. Their reaction may be a list of actions or even no action. The reaction may cause changes in the variables. For example, the reaction may generate some changes in the values of the status and/or relationship.

FIG. 14 is an example system 1400 according to the present disclosure. The system 1400 may include an engine 1401 including a persistent storage 1402 and a memory 1403. The system 1400 may include a backend 1406 and a frontend 1405. A script storage service 1407, a script crud service 1408, a frontend session storage 1409, a general object model library 1410, and a general animation library 1411 may be stored in the persistent storage 1402 for access and/or execution by one or more of the respective computer processors, usually through one or more memories of memory. Persistent storage 1402 is at least more persistent than a signal in transit, may store soft logic and/or data, may be on a tangible medium (such as magnetic or optical domains), and may be substantially less persistent than permanent storage.

Each of the script storage service 1407, script crud service 1408, frontend session storage 1409, general object model library 1410, and general animation library 1411 may include both machine readable and performable instructions and/or data of the type that may be stored in a database. The persistent storage 1402 may include a solid state hard drive, a semiconductor storage device, read-only memory (ROM), erasable programmable read-only memory (EPROM), flash memory, or any other computer-readable storage media that is capable of storing program instructions or digital information.

The media used by the persistent storage 1402 may be removable, such as a removable hard drive, optical and magnetic disks, thumb drives, and smart cards that are inserted into a drive

for transfer onto another computer-readable storage medium that is also part of the persistent storage 1402.

In the memory 1403, there may be a script processor and filters 1412, a NLP module 1413, a script text validity checking unit 1417, a script analyzing service 1418, and event loop 1422. The NLP module 1413 may include an intent recognition unit 1414, a sentiment recognition unit 1415, and an objects/relation extraction unit 1416. The script analyzing service 1418 may include an action data extraction unit 1419, a script parser 1420, and an entity extraction unit 1421. The memory 1403 may also include various character logic services 1423-1425.

The frontend 1405 may include a VR headset 1426. The VR headset 1426 may include an observation system 1427, an event generation system 1428, a voice recognition unit 1429, a motion mapping unit 1430, and an input device 1430. The backend 1406 may be a client end, such as on an iPad Pro™. The backend 1406 may include a customized models importing unit 1432, a customized texture/effects importing unit 1433, a unit for supervised learning and generation of new scripts 1434, a real-time preview rendering script editor 1435, a chaos box editing unit 1436, a character editing unit 1437, a scene editing unit 1438, and a script importing and editing unit 1439.

These units may communication with each other. The VR headset 1426 may communicate with the memory 1403. The real-time preview rendering script editor 1435 may communicate with the memory 1403 and/or the script crud service 1408. The script importing and editing unit 1439 may communicate with the memory 1403. The memory 1403 also may communicate with the persistent storage 1402.

FIG. 15 is a high-level component diagram of an example computing device 1500 performing the present disclosure or any portion thereof as described herein. Any of the engine, backend, and frontend may be performed by and/or implemented on the computing device 1500. The computing device 1500 may include one or more interconnected nodes 1510A-D. Each node 1510A-B may in turn include one or more physical processors (e.g., CPU 1520A-C) communicatively coupled to memory devices (e.g., MD 1530A-C) and input/output devices (e.g., I/O 1540A-B). Each node 1510C-D may include a hardware device 1550A-B. In an example embodiment, a hardware device (e.g., 1550A-B) may include a network device (e.g., a network interface controller (NIC), a network adapter, or any other component that connects a computer to a computer network), a peripheral component interconnect (PCI) device, storage devices, sound or video adaptors, photo/video cameras, printer devices, keyboards, displays, etc. Memory devices 1530A-C may include a volatile or non-volatile memory device, such as RAM, ROM, EEPROM, or any other device capable of storing data. As discussed herein, I/O devices 1540A-B may include any device capable of providing an interface between one or more processor pins and an external device capable of inputting and/or outputting binary data.

Processors 1520A-C may be interconnected using a variety of techniques, ranging from a point-to-point processor interconnect, to a system area network, such as an Ethernet-based network. Local connections within each node 1510A-D, including the connections between a processor 1520A and a memory device 1530A-B and between a processor 1520A and an I/O

device 1540A may be provided by one or more local buses of suitable architecture, for example, peripheral component interconnect (PCI). As used herein, a device of the host OS (or "host device") may refer to CPU 1520A-C, MD 1530A-C, I/O 1540A-B, a software device, and/or hardware device 1550A-B. Although the computing device 1500 comprises multiple nodes 1510A-D, some examples may omit one or more of the nodes 1510A-D. Some examples, may utilize a single CPU, memory device, or I/O device.

The computing device 1500 may run multiple virtual machines, by executing a software layer (e.g., hypervisor 1580) above the hardware and below the virtual machines. In some examples, the hypervisor 1580 may be a component of a host operating system (OS) executed by the computing device 1500. In another example embodiment, the hypervisor 1580 may be provided by an application running on the host OS, or may run directly on the computing device 1500 without an operating system beneath it. The hypervisor 1580 may virtualize the physical layer, including processors, memory, and I/O devices, and present this virtualization to virtual machines as devices, including virtual processors, virtual memory devices, and/or virtual I/O devices, etc.

In some examples, a virtual machine may execute a guest operating system which may utilize the underlying virtual processors, virtual memory devices, and virtual I/O devices. One or more applications 1598 may be running on a virtual machine under the guest operating system. The virtual machine may include a device register. As used herein, a device register refers to a configuration space of a device. In an example embodiment, a device may be a device of a guest operating system (that is, a "guest device"). In an example embodiment, the device register is a guest device register. In an example embodiment, a device register may be a command register or a base address register (BAR). In an example embodiment, a device register may include any known register used in the peripheral component interconnect (PCI) configuration space. In an example embodiment, a base address register (BAR) may include a base (or start) address of the guest device at which a memory region of the host device (which corresponds to the guest device) can be accessed and further includes a size indicator which denotes the size of the memory region of the host device.

In some examples, a virtual machine may include multiple virtual processors. Processor virtualization may be implemented by the hypervisor 180 scheduling time slots on one or more physical processors 1520A-C such that from the guest operating system's perspective those time slots are scheduled on a virtual processor. In some examples, a virtual machine may include virtual devices. A virtual device may provide the functionality of traditional hardware devices such as network devices, PCI devices, storage devices, sound or video adaptors, photo/video cameras, printer devices, keyboards, displays, etc.

Any of nodes 1510A-B and/or any virtual machine described herein may be depicted as the example computing device 1600 in FIG. 16 in more detail. The computing device 1600 may include a logical processor 1602, e.g., an execution core. The computing device 1600 may have one logical processor 1602 as illustrated or multiple logical processors, e.g., multiple execution

cores per processor substrate and/or multiple processor substrates each having multiple execution cores.

Communications between various components of the computing device 1600 may be provided by any architecture designed for passing data and/or control information between processors (such as microprocessors, communications and network processors, etc.), system memory, peripheral devices, and any other hardware components within a system. For example, one or various computer readable storage media 1610 may be interconnected by one or more system buses which couples various system components to the logical processor 1602. The system buses may be any of several types of bus structures including a memory bus or memory controller, a peripheral bus, and a local bus using any of a variety of bus architectures.

The computer readable storage media 1610 may include, for example, random access memory (RAM) 1604, storage device 1606, e.g., electromechanical hard drive, solid state hard drive, etc., firmware 1608, e.g., FLASH RAM or ROM, and removable storage devices 1618 such as, for example, CD-ROMs, floppy disks, DVDs, FLASH drives, external storage devices, etc. Other types of computer readable storage media, such as magnetic cassettes, flash memory cards, digital video disks, and Bernoulli cartridges, may also be used.

The computer readable storage media may provide non-volatile storage of executable instructions 1622, data structures, program modules, and other data for the computing device 1600. A basic input/output system (BIOS) 1620 that helps transfer information within the computing device 1600, such as during startup, may be provided in firmware 1608. Any of the data, modules, services, libraries, units, and/or systems of the present disclosure described herein may be stored on firmware 1608, storage device 1606, RAM 304, and/or removable storage devices 1618, and executed by logical processor 1602 or ASICs.

Commands and information may be received by the computing device 1600 through input/output devices 1616 which may include, but are not limited to, a keyboard and pointing device, a microphone, joystick, game pad, scanner or the like, a touch screen, a sensor, and/or combinations thereof. These input devices may be connected to the logical processor 1602 through a serial port interface coupled to the system bus, a parallel port, game port, universal serial bus (USB), and/or some other interface(s). A display device 1618 may provide a mechanism to display data to a user and may be, for example, a computer monitor or a smart phone display screen. The display device 1618 may also be connected to the system bus via an interface, such as a video adapter which may be part of or connected to a graphics processor 1612. There may be other peripheral output devices (not shown), such as speakers. The computing device 1600 may also include a host adapter, Small Computer System Interface (SCSI) bus, and an external storage device connected to the SCSI bus.

Turning back to FIG. 15, in some examples, a middleware 1560 or 1560A-B may run on an operating system (OS) 1587. In some examples, the OS 1587 may be a host OS. In some other examples, the OS 1587 may be a guest OS. The code that a middleware 1560A-B runs on the OS may utilize the resources of the host OS, such as the memory devices 1530A-C, and the input/output devices 1540A-B as well as the resources of one or more of the virtual machines

including the virtual processors, the virtual memory, and the virtual I/O devices. The middleware 1560A-B may be integral to information technology based on Extensible Markup Language (XML), Simple Object Access Protocol (SOAP), Web services, SOA, Web 2.0 infrastructure, and Lightweight Directory Access Protocol (LDAP). In some examples, the middleware 1560A-B may be a web server 1561, an application server 1562, a content management system 1563, or other similar tools that support development and delivery of application 1598.

In some examples, the middleware 1560A-B may include different components, for example, messaging 1564, web connector 1565, in-memory cache 1566, logging 1567, database access 1568, and/or http 1569. These components may not originally be designed to be integrated together. They may run independent of each other or be integrated together having unified configuration and runtime experience. In some examples, the middleware 1560A-B may include the component of logging 1567. Logging 1567 may be an easy way for a developer of the application 1598 to trace what has been happening during operation of the application 1598 in a log file. In some examples, the developer can find problems in the application 1598 based on the log file.

The system 1400 may be effectuated by different physical architectures than the ones depicted. For example, the system 1400 could be effectuated by distributed architecture where different physical computer systems spread over a large area, e.g., the world, are configured to perform the various herein described functions. In addition, one of skill in the art may appreciate that a single computer system could effectuate the entire system 1400. The system 1400 may be effectuated by server side and client side components.

In various examples, as shown in FIG. 17, the computing device 1700 (1400, 1500, or 1600) may operate in a network 1790 using logical connections to one or more computing devices 1701, 1702, 1703, 1704, ... The computing devices 1701, 1702, 1703, 1704, ...each may include many or all of the elements described above relative to the computing device 1700 and each may be a laptop computer, tablet computer, netbook computer, personal computer (PC), a desktop computer, a personal digital assistant (PDA), a smart phone, or any programmable electronic device capable of communicating with another computing device via network.

Network may be, for example, a local area network (LAN), a wide area network (WAN) such as the Internet, or a combination of the two, and may include wired, wireless, or fiber optic connections. In general, network may be any combination of connections and protocols that will support communications between server and user systems. When used in a network, computing device 1700 (e.g., 1600) may be connected to a LAN or WAN through a network interface card (NIC) 1614 that may include its own processor. The NIC 1614, which may be internal or external, may be connected to the system bus. In the network, any program module depicted relative to the computing device 1600 may be stored in a remote memory storage device.

Each of the computing devices 1701, 1702, 1703, 1704, ... may comprise various hardware including one or more processors 1708a, 1708b, and a data storage device 1740. As used herein, processor or processors 1708a, 1708b may refer to a device capable of executing instructions encoding arithmetic, logical, and/or I/O operations. In some examples, a processor may follow

the Von Neumann architectural model and may include an arithmetic logic unit (ALU), a control unit, and a plurality of registers. In a further aspect, a processor may be a single core processor which is typically capable of executing one instruction at a time (or process a single pipeline of instructions), or a multi-core processor which may simultaneously execute multiple instructions. In another aspect, a processor may be implemented as a single integrated circuit, two or more integrated circuits, or may be a component of a multi-chip module (e.g., in which individual processor dies are included in a single integrated circuit package and hence share a single socket). A processor may also be referred to as a central processing unit (CPU). The data storage device 1740 may be any suitable type of data storage device including, for example, one or more disks, one or more solid state drives, random access memory (RAM), cache memory, etc. In some examples, the data storage device 1740 may comprise multiple physical devices. In some examples, additional hardware may be included such as, for example, network adapters, various sensors, etc.

Above the hardware 1708a, 1708b, 1740, an operating system 1720 and file system 1730 may provide an interface between one or more applications 1710 and the hardware 1708a, 1708b, 1740. For example, the application 1710 may provide data requests to the operating system 1720. Data requests may include any type of request to manipulate data (*e.g.*, data stored at the data storage device 1740). Example data requests include read requests and write requests. Data requests may include a logical name referring to one or more data blocks 1714a, 1714b, 1714c, 1714d, 1714n to be read or written and an operation to be performed on the data block. When the data request is a write request, it may also include a data block or blocks to be written to the data storage device 1740. The logical name, for example, may refer to a file, directory, *etc.*, or other logical grouping, which may be defined by the file system 1730 of the relevant data storage device 1740. In some examples, the logical name may be a logical name of a data block or data blocks. The operating system 1720 and/or the file system 1730 may identify one or more of the data blocks 1714a, 1714b, 1714c, 1714d, 1714n referenced by the logical name and perform the requested operation.

Data blocks 1714a, 1714b, 1714c, 1714d, 1714n may be units or blocks of data below the logical or file level. For example, logical constructs, such as files, directories, metadata, etc., may include one or more of the data blocks 1714a, 1714b, 1714c, 1714d, 1714n. In some examples, data blocks are sized to correspond to the smallest unit of data handled by the data storage device 1740, which may depend on the physical device or devices making up the data storage device 1740 and/or the organization of the file system 1730. In some examples, data blocks 1714a, 1714b, 1714c, 1714d, 1714n may correspond to physical sectors at the data storage device 1640 or its subcomponents. For example, some physical data storage devices use sectors that are 4 kilobytes. Some data storage devices may use slightly larger sectors, for example, including additional bytes that may be used for a checksum. Other example sector sizes include 512 bytes, 1024 bytes, *etc.* Accordingly, some examples may utilize data blocks 1714a, 1714b, 1714c, 1714d, 1714n that are 512 bytes, 1024 bytes, 4 kilobytes, or any other suitable size. Also, in some examples, data blocks 1714a, 1714b, 1714c, 1714d, 1714n may correspond to clusters of the file system 11. For example, a typical file system cluster may be 4096 bytes or 4 kilobytes (kB) and, some physical storage

devices, such as CD-ROM's, have clusters that are 2048 bytes (2 kB). Accordingly, 4 kB and 2 kB data blocks may be used in some embodiments.

A system according to the present disclosure may be configured to implement a movie distribution network 1790 that that allows users to broadcast the movie they produced. The distribution network 1790 may include a media server that may stream content to IP addresses associated with computing systems such as a mobile device 1704 or computer system 1701. The distribution network 1790 may include a social networking engine. The social networking engine may be configured to effectuate an interactive community of user made cast members for a particular show that is broadcasted on the distribution network 1790 and/or a publishing engine to make the movie available to the public.

The following are various example elements of the present disclosure, including example data structures, variables, etc.

1 Overall Story Control

1.1 Story Block

In various examples, although Chaos Box is used to create almost infinite possibilities for the scenes in a movie, the scale of the overall storyline may be controlled. If the storyline becomes too diverging, the experience may not be so intense for the viewer. Story Block may be used to control the storyline, and each Story Block may have only one input, multiple outputs, and one or more Chaos Boxes. Story Block mostly bases on different scenes.

The following is an example Story Block:

```
"block":{
"input": null,
"name": "begin",
"output": {
        "branches": [{
                "next": "block_1",
                "events": [{
                        "from_entity": {
                                "type": "specific",
                                "id": 1234
                        },
                        "target_entity": {
                                "type": "specific",
                                "id": 1234
                        },
                        "event_type": "hurt/speak/crash",
                        "event_context": {}
                }],
                "status": [{
                        "entity": 1,
```

```
                    "health": {
                        "$eq": 100
                    },
                    "mood": {
                        "$lt": 0.40
                    }
                }]
            }]
        }
}
```

The "input" may represent the previous input Story Block's name. If this Story Block is the initial Story Block, then input may be null. The "name" may be the unique identity of each Story Block. The "output" may be the most important part defined in a Story Block's data structure. It may contain one or more branch objects. The branch objects may define the activation requirements and the information for the next Story Block. Activation requirements may include triggers and notes, such as event triggers and status triggers. Different activation requirements that lead to the same Story Block may be stored into separate branch objects.

Some events may be caused by actions from the viewer or any NPC. The events may be directly triggered or indirectly through an action-event chain-reaction. The events may be triggered in the order that matches a stored order.

Branches may be triggered by some NPC status. For example, when some NPC gets mad (e.g mood is below a certain value), this Story Block will be ended and enter a new Story Block. A lot of the times, a status trigger may be accompanied by an event trigger.

1.2 Chaos Box

As shown in FIG. 18, each Chaos Box 1800 may have one input 1801 and one or more outputs, e.g., multiple branches 1802-1804.

The Chaos Box may have a time limit since a short period of story may not happen forever. For example, when a bank robber is pointing a gun at someone and asking questions, there is usually a time limit for this situation.

In the following example, UUID may be used to identify different Chaos Boxes. The "time_limit" (in second) may be the longest time for the player to experience the whole Chaos Box. If the time limit has been reached, then the user may ask the server for default action that should be taken under the current status in order to push the story forward and disable user input until the Chaos Box has reached its output. If the time limit is set to null, then it may allow the players to play as long as they want until they find an exit point for the scenario.

```
"box": {
"input": "[UUID]",
"id": "[UUID]",
"block": "[block_name]",
"time_limit": 180,
```

```
"output": {
        "branches": [{
                "next": "[UUID]",
                "events": [{
                        "from_entity": {
                                "type": "specific",
                                "id": 1234
                        },
                        "target_entity": {
                                "type": "specific",
                                "id": 1234
                        },
                        "event_type": "hurt/speak/crash",
                        "event_context": {}
                }],
                "status": [{
                        "entity": 1,
                        "health": {
                                "$eq": 100
                        },
                        "mood": {
                                "$lt": 0.40
                        }
                }]
        }]
}
}
```

Also, there may be lots of box constraints when the viewer/player is interacting in a Chaos Box. These constraints may not be stored with the box objects, but may be stored separately and become a part of the Logic Processor when it starts to process the basic unit inside a certain NPC by some certain events.

2 Data Structure

2.1 Entity

An NPC may be defined as an entity. The entity may include any objects that are capable of reacting to an input. The entity objects only include basic information. The following is an example:

```
{
 "id":1,
 "name":"Paul Grand",
```

```
 "description":"A mysteric man."
}
```

2.1.1 NPC/entity Status

Each NPC/entity may have a status including some attributes. This whole structure may be included in the session.

```
        "npc":{
 "entity":1,
 "health":100,
 "mood":0.43,
 "position":{
  "x-axis":"1003.444",
  "y-axis":"433.223",
  "z-axis":"32.21"
        },
 "movespeed":0.88
}
```

In the above example, the entity's status includes "health," "mood," "position," and "movespeed." "Health" may be the basic attribute of the NPC and represent the health value. The value of this attribute may decide all the actions responding to different event. Its value may be an integer value between 0 and 100. "Mood" may be an attribute representing a simple positive/negative status of the NPC. The default value may be 0.50 and may have 2 decimal places. When the value is below the default value, the lower the more negative it may represent. When the value is above 0.50, the higher the more positive it may represent. Position may also be one of the attributes. The precision of the value of each axis may be 3 decimal places. "Movespeed" may also be one of the attributes and represent the ratio of the current movement speed of the NPC to a normal speed. It may decrease due to some injury or may increase due to some special buffs. Its range may be between 0.00 and 2.00.

2.1.2 NPC/entity Relationship

The following is an example data structure of an NPC/entity's relationship:

```
        "relationship":{
 "from_entity":1,
 "target_entity":2,
 "teamspirit":0.45,
 "hostility":0.05,
 "friendness":0.80
}
```

The relationship may be mono-directional, and there may be only one relationship data with the same from and target entity. The "from_entity" may be the source entity of the relationship. "Teamspirit" may represent the co-operation senses and may have a value between 0.00 and 1.00. If this value goes higher, when the target entity chooses to make some important

decision or take actions such as attacking some other entities or moving to some other positions, the source entity will be more likely to follow or join the target entity. This may be decided in the Logic Processor within different basic units. “Hostility” may represent how the source entity treats the target entity as enemy and may have a value between 0.00 and 1.00. If this value goes higher, when the target entity chooses to do some aggressive actions or cause some aggressive event (e.g the target entity crashed into the source entity), the source entity may be more likely to act back with some aggressive words or actions towards the target entity. On the other hand, if this value is low enough, the source entity may be more likely to ignore or take it easy, but each time the hostility may increase a little bit if the target entity continues to attack some other entities with words or a physical object in the virtual word. “Friendness” may be the opposite attribute from hostility. It may also have a value between 0.00 and 1.00. All these relationships may be stored in the Session Context and may be changed by some basic units along with the output of a list of responded actions.

2.2 Object

Each object available to interact may be included in the Session. The following is an example data structure of an object:

```
"object":{
 "id":1,
 "title":"Glock pistol",
 "description":"A basic weapon",
 "owner_entity":0,
 "usage":"attack",
 "position":{
  "x-axis":"1003.444",
  "y-axis":"433.223",
  "z-axis":"32.21"
 }
}
```

For example, a pistol's usage may only be attack, but the usage of a handle on a container may be open, close, or grab (then throw it to any direction through client's physic engine). The “owner_entity” may represent the object's owner's information. If the object has no owner and is free to use, then this attribute may be set to have a value of 0.

In different Chaos Boxes, there may be different objects. The positions of these objects may be included and updated by the client physic engine and synchronized with the server. For example, if the user happened to activate an exit of a certain Chaos Box where, for example, the user blew up the car used for running away, then in the next Chaos Box, the user may only run away on foot.

2.3 Event

2.3.1 Basic Data Structure

All events may be produced by the physic engine on the user's end. Special event data structure may be used in the basic unit. The following is an example:

```
"event":{
 "from_entity": {
  "type":"specific/fuzzy",
  "id":1234
 },
 "target_entity":{
  "type":"specific/fuzzy",
  "id":1234
 },
 "event_type":"hurt/speak/crash",
 "event_context": {}
}
```

The "event_context" may have different data structures according to different "event_type." There may be an extra attribute "observe_entity" that may be produced from the client's side. It may represent the entity that is observing this event. It may be the same entity as the target_entity, but may not be the same as the from_entity.

2.3.2 From Entity

In the process of the movie, this parameter from_entity's type may always be specific, since every event happened in the virtual world may have a specific source. When the event is not caused by any other NPC than the target entity itself, the from_entity may be 0. In the basic unit, from entity's type could be fuzzy, which means it could use a value filter to match the actual entity given in the running procedure. In the fuzzy mode, the filter may be specified using direct statement as in the following example:

```
"from_entity":{
 "type":"fuzzy",
 "id":0,
 "teamspirit":{
  "$gt":"0.7"
 },
 "friendness":{
  "$gt":"0.7"
 },
 "hostility":{
  "$gt":"0.7"
 }
}
```

When the attribute is not stated here, it may mean that there is no filter to this attribute. The filters could include '$gt', '$lt', and '$eq', which represent greater than, less than, and equal respectively. In the fuzzy mode, all the npc_relationship related attributes may be stated prominently. If one attribute has no range limitation, it may be stated as "$gt":"0.0." If the id is specific, it may not be the entity id of the basic unit owner.

2.3.3 Target Entity

The usage of the target entity may almost be the same as the from_entity. However, the target entity may be the basic unit's owner when specified. In some occasions, the target_entity may be the from_entity itself, such as the user shooting at themselves or saying some words to themselves. The other NPC may react towards this event.

2.3.4 Event Type

The following are some example event types:

- Hurt: This may be caused by an Attack action. If the hurt's value causes the health to decrease to 0, the server may respond with an action of "die" and generate a death event instead of hurt.
- Speak: This may be caused by an action of speaking.
- Crash: This may be caused by an action of moving.
- Falldown: This may be caused by another NPC or the NPC itself. If the cause is the NPC itself, the from_entity may be 0.
- Dead: This may be caused by an action of dying.

2.3.5 Event Context

Different event types may have different data structures of event_context.

Hurt:

```
"event_context":{
 "object_id":10,
 "value":20
}
```

For example, object X may be used to reduce the value of health to a certain value. In the virtual world, the operating mechanism may refer to the real world. Thus, once an NPC gets shot, except by healing with medicine, going to a hospital, or being reset by the overall scenario engine (e.g the story progressed to a year after the current scenario), the NPC may not increase its health value automatically or by some items as in some video games.

Speak:

```
"event_context":{
 "content":"None of your business"
}
```

A normal action of speaking may cause a speak event. If the speak action has no target, the target entity of this event may be the source entity itself. This event may be produced from the client's end when any NPC or the user inputs a speech.

There may be additional fields in the basic unit. For example,

"event_context": {
"content":"None of your business",
"intent":"assult",
"level":0.8,
"key_entity":"",
"dialog": "waiting_apology",
"dialog_state": "initial"
}

This version of event_context of speak may be designed for the basic unit. A normal speak action may not produce this type of data structure, which may be used after NLP and Intent Recognition. Attributes, such as intent, level, and key_entity, may all be generated after the NLP process.

Crash:

"event_context":{
"range":"far/normal/close",
}

The range may represent the distance from the place that the crash happened to the position of the basic unit owner. It may have three optional values, such as “far,” “normal,” and “close.” When the logic owner is the target itself (it got crashed by some other NPC), then the range here could be “close.”

Falldown:

"event_context":{
}

The “falldown” may have no event context, but it may be prior to the crash event if the falldown happened right after the crash event. For example, if a crash event happens to an NPC, the client's physic engine may wait till no falldown event is generated or a falldown event is generated before uploading the event.

2.4 Action

To each event, the response could be one action or a list of actions.

2.4.1 Basic Data Structure

Special action data structures may be used in the basic unit. The following is an example of the basic data structure of an action.

"action":{
"action_type":"speak/use/move/shock/default/die",
"from_entity":1,
"target_entity":2,
"action_context":{}
}

2.4.2 Action Type

Actions may be generated from the server side and may be executed on the client side. The action type list may be synced between the client and the server. If the server is going to support a new action type, it may wait after the upgrade of the client side to also include the new action type. Alternatively, the user may handle this situation when receiving an unknown action type.

The following are some example action types:

- speak, say something
- use, use some available objects to open/close/grab/use it to attack other entities
- move, move randomly (panic status) or close to/away from the target entity
- shock, just get shock
- die, start dying animation
- default, do nothing but keep idle status

2.4.3 Action Context

Different action types may have different data structures of action_context. For example,

Speak:

```
"action_context":{
"content":"none of your business "
}
```

When server is responding to an action, the speak action may be as simple as above right now. In the basic unit, the speak action may have two more fields:

```
"action_context":{
"content":"none of your business",
"dialog": "waiting_apology",
"dialog_state": "initial"
}
```

The dialog and dialog_state may represent the next state of dialog mode of the current session.

Use:

```
"action_context":{
"object":2,
"usage":"open/close/grab/attack",
"strength":50
}
```

The objects available may be the same in one Chaos Box, but may also be limited for different actions that target the user.

Move:

```
"action_context":{
"type":"random/targeted",
"target_entity":2,
"direction":"close/far",
```

"speed":"slow/normal/fast"
}

If the type is random, then there may be no need to specify the target_entity and the direction. If the type is given targeted, then there may be target_entity and direction to specify the target direction.

Shock & default & die:

"action_context":null

There may be no context for shock and default action. Under some circumstances, although default actions may not have any visual actions from the outside, there may still be some changes within the NPC status in this session, e.g increased mood value.

2.5 Session

The session may be an important data object and may contain all the status, relationships of all the NPCs and all the available objects data in the current scene. The session may be synchronized between the server and the user all the time.

The npc_status and the npc_relationship may be updated on the server side. The current_available_objects may be updated on the client side since the movement of the objects may be executed inside the client's physic engine, so the user may tell the server the current situation when the NPC needs to react through logic processor.

The user may upload a previous session when a new event is generated by the client's physic engine, whether the NPCs or the user caused this event.

The session may be created when a new movie starts. Each time the server may generate a unique session_id for the new session. The user may request the initial session before any possible first event.

The following is an example session's data structure:

```
"session":{
"session_id": "[UUID]",
"current_observe_entity": 2,
"current_block": "bank_hall",
"current_box": "[UUID]",
"session_context": {
        "npc_status": [{
                "entity": 1,
                "health": 100,
                "mood": 0.43,
                "position": {
                        "x-axis": "1003.444",
                        "y-axis": "433.223",
                        "z-axis": "32.21"
                },
                "movespeed": 0.88
```

```
        }],
        "npc_relationship": [{
            "from_entity": 1,
            "target_entity": 2,
            "teamspirit": 0.45,
            "hostility": 0.05,
            "friendness": 0.80
        }],
        "current_available_objects": [{
            "id": 1,
            "title": "Glock pistol",
            "description": "A basic weapon",
            "owner_entity": 1,
            "usage": [
        "attack",
        "pointto"
    ],
            "position": {
                "x-axis": "1003.444",
                "y-axis": "433.223",
                "z-axis": "32.21"
            }
        }]
    }
}
```


3 Basic Unit

All the NPCs may have a logical “brain” to process all the incoming events including events they see or hear, whether or not this event's target is themselves. If the target is not the observer itself, its logic process maybe different since reaction may be based on the relationship between the observer and the target.

During actual running of the movie, each event that has been uploaded to the server may be with a specific observer, representing using which the NPC's logic to process this event. For example,


```
    "event":{
  "from_entity": {
    "type":"specific/fuzzy",
    "id":1234
  },
  "target_entity":{
    "type":"specific/fuzzy",
```

"id":1234
},
"observe_entity":2,
"event_type":"hurt/speak/crash",
"event_context": {}
}

Each request may be a procedure that absorbs one event object and/or generate one or more action objects. This procedure may be called the Basic Unit. FIG. 19 shows an example procedure. As shown in FIG. 19, if an event 1901 is a speak event 1902, the dialog may be processed by NLP 1905. The intent and/or the key entity of the dialog may be recognized 1904. The dialog management 1905 may process the results and transmit the processed results to the logic processor 1906. On the other hand, if the event is a non-speak event 1907, the event 1901 may be entered into the logic processor 1906 directly. The logic processor 1906 may process the event 1901 and generate a chose box 1908 including a plurality of basic unit scripts 1909, 1910, 1911.... The chaos box 1908 may include box constraint(s) 1912. The chaos box 1908 may then generate output action(s) 1913 as the outcome.

3.1 Logic Processor

Logic Processor is a service to process the input post-processed event data. As shown in Fig. 19, described above, for a non-speak event, the event context may be directly send to the logic processor; for a speak event, the content may go through the NLP service first and generate the most possible intent and key entity inside the conversation. The following are three example matching patterns for NLP:

Intent Recognition: Intent may be objective, so it may not be related to any characteristics. When recognizing intent, there may be a general recognizer and/or a specific recognizer related to the observer entity and the Chaos Box this session is currently in. The recognizer may be trained using pre-marked data. For example, in a movie where a bank robber is entering a bank, at this Chaos Box inside the bank hall scene or Story Block, whatever the robber says to the security guard, the conversation may go through an intent recognizer to check if the robber is asking the security guard to join him and rob the bank together. The necessary information may be extracted from previously defined plots. This kind of specific recognizer may be reused by some other movies.

Keyword Match using Dictionary: In some Chaos Box, the NPC may react to certain words, places, NPCs and/or items. It may be the trigger of some side story. This may be configured in the Logic script.

Q&A: This may mainly be used for some user guidance or whenever the user does not know how to continue. If the user says something that matches the Q&A list, the NPC around the user may turn into the guidance mode and teach the user how to proceed. This matching pattern may be disabled after several story blocks assuming the user will know exactly how to proceed after some interactions.

3.2 Basic Unit Script (sometimes referred to as Logic Script)

Basic Unit Script may be specific and may mainly work for the story. It may also be fuzzy and may work for the infinite possibilities in a Chaos Box. The following is an example.

```
{
"owner_entity": 1,
"chaos_box": 1,
"status": {
        "mood": {
                "$lt": "0.7",
                "$gt": "0.3"
        },
        "health": {
                "$gt": "60"
        }
},
"other_status": {
}
"event": {
        "from_entity": {
                "type": "fuzzy",
                "id": null,
                "teamspirit": {
                        "$gt": "0.7"
                },
                "friendness": {
                        "$gt": "0.7"
                }
        },
        "target_entity": {
                "type": "specific",
                "id": 1
        },
        "event_type": "hurt",
        "event_context": {
                "object_id": 2,
                "value": 10
        }
},
"action": [{
        "action_type": "speak",
```

```
                "from_entity": 1,
                "target_entity": 2,
                "action_context": {
                        "content": "Watch out, bro!",
                        "dialog": "waiting_apology",
                 "dialog_state": "initial"
                },
                "current_box": 1
        }],
        "value_effect": {
          "status": {
            "mood": "0.3",
            "health": "0",
            "movespeed": "0.1"
          },
          "relationship": [{
            "type":"specific",
            "from_entity":2,
            "target_entity":1,
            "teamspirit":"0.1",
            "friendness":"0.0",
            "hostility":"0.0"
          },{
            "type":"fuzzy",
            "direction":"from/target",
            "teamspirit":"0.1",
            "friendness":"0.0",
            "hostility":"0.0"
          }]
        }
}
```


For example, the NPC with entity id 1 may be a person with the name Bob. This example represents a logic owned by Bob. He seems to be a nice and gentle guy. This basic unit may be defined in the Chaos Box which has id 1 and may also be widely reused since this is a very general logic for Bob. When a person hurt Bob using a stick (assuming a stick is an object with id 2) and caused little damage, he would only say “Watch out, bro!” and enter a dialog mode where he is waiting for this person’s apology. If the next word this person says is not an apology, Bob will respond directly from the Dialog Management service and cause a little decrease in the “friendness” and “teamspirit” in the “npc_relationship” with this person, as well as Bob’s mood in the “npc_status.”

After several rounds, due to more and more decreases in the “friendness” between Bob and the person, this person’s behavior will be processed by another Basic Unit Script which is responsible to handle all the hurt events from those who have lower “friendness” with him. In that case, Bob may hurt this person back using whatever objects near him that he may use to attack this person.

All these basic units of Bob may be modified on the server at runtime. Generally, the logic pattern of each character under a current world view may be defined. The logic scripts may be divided into different parts, for example,

Common Sense Logic: This part may be highly re-usable and may need only slight change to suit a new world view.

Characteristic Logic: For example, if Bob is a nice and gentle guy, then he will react to some events nicely. Then, if Joseph is also a nice and gentle guy, he may directly re-use the “nice and gentle” part of Bob.

Story Limited Logic: This part may be designed and written specifically for each story.

3.3 Box Constraint

The storyline may be trapped because of some actions that seem logical but in fact will break some important node(s) in the storyline. For example, if an angry NPC killed a man who is necessary for the rest of the story by mistake, the story may be trapped. Especially, when the scale of each Chaos Box grows larger, there may be the need to define a set of box constraints for all the NPCs' actions inside a Chaos Box.

3.4 Dialog Management

Dialog Management (DM) may be used for some multi-round conversations, especially conversations that are essential to the story. For example, in a bank robbery scene,

- The bank robber may ask the user "What's the passcode to the vault?" Then the DM may set the dialog mode into “asking_passcode”;
- If the user says "I don't know" or "It's 1123," then this conversation may be over and proceed to next action;
- If the user says something insulting, it may also be valid since this may irritate the robber and lower his mood and increase his hostility to the user, but he may continue to ask the user. If the user continues to insult him, then the robber may shoot the user in the end;
- If the user says "what's the weather like today" or anything else that is irrelevant to this dialog, the robber may be triggered almost the same as above. For example, this may be viewed as “insulting” by the robber, so if the user continues to say things apparently outside this dialog or scenario, this dialog will not end, but the story might be ended. At a lot of times, a dialog may be closed after several rounds of irrelevant conversations, depending on the dialog itself.

DM may be needed only when the action_type is “speak” and the target_entity is the entity that user acts as. What happens between NPCs may be all defined in the Logic Script.

When uploading an event, the client's physic engine may be responsible for the update of the curren_available_objects, for example, the position update or the availability check. This may provide important environment information to the server when making Logic Processing. The server may update other information such as npc_status, npc_relationship and current_box during the Logic Processing. The user may maintain a session from the beginning. Every time the user is going to request the server for event uploading, the session object may be latest-updated.

4. User Interface

FIGS. 20A-20Z show example user interfaces according to some aspects of the present disclosure performing the steps in FIGS. 5-9. When a user opens an example application for the present disclosure, an initial screen (FIG. 20A) may prompt the user to click on the "+Create an Interactive Movie" button. Then the screen may show the buttons for all the steps the user needs to go through to create the movie as shown in FIG. 20B. Only the button for the current step may be usable. The user may click the Step 1 (upload your script) button and enter a folder where existing scripts are stored (FIG. 20C). The user may choose any of the scripts and upload it (FIG. 20D). After the selected script has been uploaded, the user may click the Step 2 (check and edit scene) button (FIG. 20E). FIG. 20F shows an example scene, a bank. The user may select the portion of the script related to the scene (FIG. 20G). The scene then may be uploaded (FIG. 20H). The user now may start Step 3 (check and edit NPC's character) (FIG. 20I) and may be presented with the cards for all available characters (FIG. 20J). In this step, the user may also create new characters by adding new cards for the characters. Then the selected characters may be uploaded (FIG. 20K). The user may now create branches in Step 4 (FIGS. 20L-20O). After the branches have been uploaded (FIG. 20P), the user may review animation and do rendering in Step 5 (FIGS. 20Q and 20U) and upload the animation and rendering (FIG. 20V). The user may finish and publish the movie in Step 6 (FIGS. 20W-20Y) after entering the name and description of the movie and choosing the VR platform (FIG. 20X). In the end, the screen may show the message that the user's application has been submitted and also offer the option of republish (FIG. 20Z).

5. Examples of different users using a portion or portions of the present disclosure.

5.1 Frontend User

A viewer may want to move the direction of their view freely, so that they can feel like they are the character in the movie.

A viewer may want to do whatever they are allowed to do such as speaking or using available items in the movie, so that they can feel like they are the character they are acting.

A viewer may want to get responses from other characters if they speak or do something to tose characters, so that the viewer can interact with them.

A viewer may want to be involved in some drama if they speak or do something to other characters, so that they can actually be part of the story.

A viewer may want to hear different voices if other characters are speaking to they, so that they can feel like the character they are acting.

A viewer may want to be implicitly told or implicitly guided to do something, so that they can live through the whole story.

5.2 Backend User

For example, in the script initializing module:

A screenwriter may want to use their existing script to generate Story Tree, so that they can write mo-script based on the story they already had.

A screenwriter may want to use their existing script to generate Logic script of different characters, so that they can write mo-script based on the story they already had.

A screenwriter may want to use their existing script to generate character profiles, so that they can write mo-script based on the story they already had.

A screenwriter may want to use their existing script to generate the existing valid actions in different scenes, so that they can use them in the later writing.

A screenwriter may want to add a story block in the Story Tree, so that they can write more possibilities for the story.

A screenwriter may want to delete a story block in the Story Tree, so that they can remove the story block they do not like.

A screenwriter may want to modify a story block in the Story Tree, so that they can change the content if that is not what they want.

A screenwriter may want to modify a character's profile, so that they can fix some error caused by auto-generation.

A screenwriter may want to add a character's profile, so that they can import new character into the story.

A screenwriter may want to delete a character's profile, so that they can remove the character they do not want.

In the script writing module:

A screenwriter may want to add a basic unit for a character in the tree mode, so that they can define what actions will this character react under certain circumstance.

A screenwriter may want to delete a basic unit for a character in the tree mode, so that they can define what actions this character will take under certain circumstances.

A screenwriter may want to modify a basic unit for a character in the tree mode, so that they can define what actions this character will take under certain circumstances.

A screenwriter may want to add input events for the logic they are writing, so that they can define the trigger events of this logic.

A screenwriter may want to delete input events for the logic they are writing, so that they can replace with another events trigger if they need to.

A screenwriter may want to write the logic down instead of dragging and dropping in the tree mode, so that they can write the story in the way they are familiar with.

A screenwriter or a designer may want to modify the status filter of the logic owner character for the logic they are writing, so that they can define under what circumstances the logic owner character will make such decisions.

A screenwriter or a designer may want to modify the status filter of the related character for the logic they writing, so that they can define under what circumstances the logic owner character will make such decisions.

A screenwriter or a designer may want to modify the relationship filter of the related characters including the logic owner itself for the logic they are writing, so that they can define under what circumstances the logic owner character will make such decisions.

A screenwriter or a designer may want to add some more filters for the logic including the key logic filters, which are the filters for checking whether some certain logics appeared in the path before, so that they can define the logic more specifically to fit the story.

A screenwriter may want to add actions for the logic, so that they can define what the logic owner would react under the circumstance they have already defined.

A screenwriter may want to see any possible events from the output actions of the logic they selected, so that they can decide which events to continue editing.

A screenwriter may want to put literally description for the logic filters, so that they can give the detailed tuning job to other detailed designers.

A numerical designer may want to read the description written by the screen writer, so that they can tune the numeric system of each filters based on the description.

A conversation designer may want to edit the input language set for the logics created by a screenwriter where the input is a speaking event, so that they can define the general intent or the key information that this logic needs, and the input does not need to be a specific sentence.

A conversation designer may want to edit the output language set for the logics created by a screenwriter where the output actions include speak actions, so that they can define the set of output content, and the output doe not need to be a specific sentence.

A screenwriter may want to see the simulated chain reaction in a preview window if they change the logic output, so that they can preview the impact of their editing.

A screenwriter may want to see the simulated chain reaction after the logic they selected happens in a preview window, so that they can go back during their writing and preview the possible consequences.

A screenwriter or a designer may want to move the position of different characters in the preview window manually, so that they can simulate different events that will be triggered by different positions of the characters.

5.3 VR player

A VR player may want to interact with the movie to create their own adventure. To do this, the VR player can either wait the end of the transit of current movie scene or choose to interact in current interaction point to the next transit, and the outcome is that the VR player may enter a new interaction point where the VR player cannot move their position freely, which may be only tiny movement around supported by the VR headset's tracker.

A VR player may be in an interaction point with an NPC asking them "do you want to go inside?" The VR player may want to tell the NPC that they want to go inside. To do this, the VR

player can speak into the microphone and say "yes!" The outcome may be that the NPC accepts the VR player's choice, and they go into the building.

A VR player may be in an interaction point and may want to move the direction of their view freely. To do this, the VR player can move their headset around, and the outcome may be that the direction of the VR player's view will be moving simultaneously.

A VR player may be in an interaction point with object "gun" available to use. The VR player may want to use the weapon to point to any characters or objects in the movie. To do this, the VR player can hold the right controller in their right hand, lift up the right hand and point to any direction they want. The outcome may be that the character's hand which holds the weapon will move synchronously with the VR player's right hand in the real world.

A VR player may be in an interaction point with object "gun" available to use. The VR player may want to trigger the weapon to fire. To do this, the VR player can pull the trigger on the controller which is reflected to the weapon in the virtual world. The outcome may be that the weapon in the virtual world will be fired.

A VR player may be in an interaction point and may want to speak to the NPC close to them. To do this, the VR player can move the direction of their view to that NPC and start speaking, and the outcome may be that the NPC will hear what the VR player said and give the VR player a response in actions.

A VR player may be in an interaction point with an NPC speaking to them. The VR player may want to hear different voice for each NPC. To do this, the VR player can speak to different NPCs and wait for their responses, and the outcome may be that each NPC will respond in different voices.

A VR player may want to get the movie tutorial in a cinematic way. To do this, the VR player can follow the guidance of a companion character called "teacher," and the outcome may be the companion character will teach the VR player how to play and how to interact in a cinematic way.

5.4 IDE user

An IDE end-user with the IDE GUI open to the home screen may want to swap the sentiment trigger associated with an existing NPC character's dialogue choice, so that "positive" and "negative" responses are flipped. To do this, the IDE user can open the IDE GUI, navigate to the existing dialogue choice detail view, click "swap" in the editor window, and then click "save." Then, the next time when the IDE user plays the VR experience, the dialogue choice is switched.

An IDE user with the IDE GUI open to the home screen may want to start creating their interactive movie. To do this, the IDE user can click the "+ Creative a Interactive Movie" button, and the outcome may be that the project will be established and the step list screen will be shown.

An IDE user with the IDE GUI open on the step list screen may want to start importing existing scripts to generate content. To do this, the IDE user can click the button "Step 1 upload your script," choose a PDF file from file storage system, and click the file. The outcome may be that it will automatically start analyzing the scripts, and the movie name will be recognized and fill their project.

An IDE user with the IDE GUI open on the step list screen may want to start importing existing scripts do generate content. To do this, the IDE user can click the button "Step 1 upload your script," choose "Dark Knight.PDF" from file storage system, and click the file, and the outcome may be that it will automatically starts analyzing the scripts, the movie name "Dark Knight" will be recognized and fill their project, and a character list of "ALEX", "QUINN", "ROOKIE", "KEANU", "BETSY", "SECURITY GUARD" "SERGEANT" will be generated.

An IDE user with the IDE GUI open on the step list screen and having just finished uploading existing scripts may want to know if there is any error when the system analyzes the scripts. To do this, the IDE user can upload a file with wrong file format or with wrong content format, and the outcome may be some messages showing on the screen about the error.

An IDE user with the IDE GUI open on the step list screen and having just successfully finished step 1 may want to check the scenes analyzed from the scripts they uploaded. To do this, the IDE user can click the button "Step 2 check and edit scenes," and the outcome may be that a new screen appears showing the list of scenes analyzed.

An IDE user with the IDE GUI open on the scene list screen may want to review or edit the scene. To do this, the IDE user can click on the scene they want to edit and enter the scene detail, and the outcome may be that the scripts text of the scene they clicked on will appear.

An IDE user with the IDE GUI open on the scene detail screen may want to read more scripts text of the current screen. To do this, the IDE user can scroll the screen, and the outcome may be that more content will be shown.

An IDE user with the IDE GUI open on the scene detail screen may want to jump to the start of the current scene. To do this, the IDE user can click the "start" button on the scene detail screen, and the outcome may be that the content will jump to the start like a Anchor Link.

An IDE user with the IDE GUI open on the scene detail screen may want to jump to the end of the current scene. To do this, the IDE user can click on the "end" button on the scene detail screen, and the outcome may be that the content will jump to the end like a Anchor Link.

An IDE user with the IDE GUI open on the scene detail screen may want to make some paragraph to an eye catch in the current scene. To do this, the IDE user can drag the green point and the red point of the current scene to the place they want; click "install eyecatch," and a blue area may appear between the green line and the red line; and click "don." The outcome may be that the eyecatch of current scene will be saved.

An IDE user with the IDE GUI open on the scene detail screen may want to delete an eyecatch in the current scene. To do this, the IDE user can click the button "remove eyecatch," and a blue area may disappear between the green line and the red line; and click "done." The outcome may be that the eyecatch of the current scene will be removed.

An IDE user with the IDE GUI open on the scene list screen may want to finish the scene and go to next step. To do this, the IDE user can click on the "confirm and build," and the outcome may be that the setting in every scene will be saved.

An IDE user with the IDE GUI open on the step list screen may want to review and edit all of the NPCs' characteristics. To do this, the IDE user can click on the "Step 3" and enter the NPC detail screen. The outcome may be that every NPC's characteristics will be shown.

An IDE user with the IDE GUI open on the NPC detail screen may want to edit a NPC's characteristics. To do this, the IDE user can click on the card they want and drag the slider bar to modify the numerical value till it is satisfactory in the floating layer. The outcome may be that the NPC's characteristics will be changed.

An IDE user with the IDE GUI open on floating layer of the NPC detail screen may want to save the change they made. The IDE user can click on the "Save" button, and the outcome may be that the NPC's characteristics will be saved.

An IDE user with the IDE GUI open on floating layer of the NPC detail screen may want to reset the change they made to the original. The IDE user can click on the "Reset" button, and the outcome may be that the NPC's characteristics will be reset.

An IDE user with the IDE GUI open on the NPC detail screen may want to add a new NPC–"John." To do this, the IDE user can click on the "+" button, click the "type name," input the name "John," and drag the slider bar to modify the numerical value till it is satisfactory in the floating layer. NPC's characteristics may then be changed.

An IDE user with the IDE GUI open on floating layer of the NPC detail screen may want to save the new NPC named John they added. The IDE user can click on the "Save" button, and the outcome may be that the characteristics of John will be established, and the NPC detail screen will include a new John's NPC card.

An IDE user with the IDE GUI open on floating layer of the NPC detail screen may want to cancel the creation of the new NPC. The IDE user can click on the "Cancel" button, and the outcome may be that the new NPC's creation will be cancelled.

An IDE user with the IDE GUI open on the NPC detail screen may want to delete one or more NPC. To do this, the IDE user can click on "select," click one or more NPC cards they want, and click on "delete." The outcome may be that the NPC cards I select will be deleted.

An IDE user with the IDE GUI open on the NPC detail screen may want to change a NPC's role from Super NPC. To do this, the IDE user can long press the card they want and move the card to somewhere below the NPC category. The outcome may be that the NPC's role will be changed.

An IDE user with the IDE GUI open on the NPC detail screen may want to finish and go to next step. To do this, the IDE user can click "confirmed and build" button, and the outcome may be that the change will be saved and new NPC will be established.

An IDE user with the IDE GUI open on the step list screen may want to review and edit the story structure. To do this, the IDE user can click on the "step 4," and the outcome may be that the story structure screen will be shown.

An IDE user with IDE GUI open on the story structure screen may want to review the structure. To do this, the IDE user can drag to move the view to somewhere they want and pinch to zoom in or out to review.

An IDE user with IDE GUI open on the story structure screen may want to review a story card detail. To do this, the IDE user can click a story card they want, and the outcome may be that the detail of this card will be shown.

An IDE user with IDE GUI open on the story structure detail may want to edit some detail. To do this, the IDE user can click the icon called "edit" on the toolbar, and the outcome may be that the story card detail will transform to edit mode.

An IDE user with IDE GUI open on the edit mode of story structure detail may want to edit some text. To do this, the IDE user can click the text below the summary or detail and input something they want. The outcome may be that the new text they input will replace original text.

An IDE user with IDE GUI open on the edit mode of story structure detail may want to change the scene that is currently displayed. To do this, the IDE user can click the drop-down to select a scene they want, and the outcome may be that the new scene they selected will replace original scene.

An IDE user with IDE GUI open on the edit mode of story structure detail may want to add a new scene named office. To do this, the IDE user can click on the "+" button beside the scene drop-down and input the name "office" for the new scene. The outcome may be that the new scene they added will replace original scene.

An IDE user with IDE GUI open on the edit mode of story structure detail may want to add one or more Main Character NPC in this story card. To do this, they can click on the button with NPC's name, and the outcome may be the state of a button will be filled, and the quantity of NPC will be changed.

An IDE user with IDE GUI open on the edit mode of story structure detail may want to delete one or more Main Character NPC in this story card. To do this, the IDE user can click the button with NPC's name, and the outcome may be that the state of a button will not be filled.

An IDE user with IDE GUI open on the edit mode of story structure detail may want to add an all new NPC. To do this, the IDE user can click on the "+" button beside name button and drag the slider bar to modify the numerical value till it is satisfactory in the floating layer. The outcome may be that the new NPC they added will be established and associated in NPC detail screen.

An IDE user with IDE GUI open on the story structure detail may want to save their change. To do this, the IDE user can click on the "done" button. The outcome may be that their change will be saved, and the story structure detail screen will disappear.

An IDE user with IDE GUI open on the story structure detail may not want to save their change. To do this, the IDE user can click on somewhere outside the detail page. The outcome may be that their change will not be saved, and the story structure detail screen will disappear.

An IDE user with IDE GUI open on the story structure screen may want to add a new story card to rich the story. To do this, the IDE user can click the icon called "add new branch" and input NPC, scene, summary and detail as described above. The outcome may be that a new story will be established.

An IDE user with IDE GUI open on the story structure screen may want to link the new card to another. To do this, the IDE user can select two cards they want to link and click the icon called "path" on toolbar. The outcome may be that the two card will be linked.

An IDE user with IDE GUI open on the story structure screen may want to link the new card to another. To do this, the IDE user can select an output to this new card input, and the outcome may be that the two cards will be linked.

An IDE user with IDE GUI open on the story structure screen may want to delete a story. To do this, the IDE user can select a card and click the icon called "delete" on toolbar. The outcome may be that the card they selected will be deleted.

An IDE user with IDE GUI open on the story structure screen may want to move a card to somewhere. To do this, the IDE user can long press the card they want and drag to somewhere they want. The outcome may be that the card will change the position to somewhere they want.

An IDE user with IDE GUI open on the story structure screen may want to switch the Main Character to review. To do this, the IDE user can click the name on the story card. The outcome may be that the detail will be changed.

An IDE user with IDE GUI open on the story structure screen may want to review and edit the reaction in a card. To do this, the IDE user can double-click the card they want, and the outcome may be that the chaos will be shown.

An IDE user with IDE GUI open on the chaos screen may want to review a NPC's reaction. To do this, the IDE user can click the card they want, and outcome may be that the reaction view will be shown.

An IDE user with IDE GUI open on the reaction view may want to add or edit a reaction. To do this, the IDE user can click the icon called "edit" on toolbar, and the outcome may be that the reaction view will be into edit model.

An IDE user with IDE GUI open on the reaction edit model may want to add a new reaction. The IDE user can click the "+ add new"; and type the object as "none," reaction as "push," target as "pickup truck," emotion as "default," and status by the preformat. The outcome may be that the input will be identified; and the demonstration animation of a model of policeman pushing a model of pickup truck will be displayed in the right window, and the truck will only be pushed forward very slowly if the strength related properties for the policeman is set higher than 50% of the value range, but if the strength related properties for the policeman is set very low, then he will not be able to push the truck forward.

An IDE user with IDE GUI open on the reaction edit model may want to add a new reaction just by traditional writing. To do this, the IDE user can click the icon "T" and type "The policeman shoots at the thief with a gun." The outcome may be that the input will be identified; and the demonstration animation of a model of policeman shooting at a model of thief with a gun in his hand will be displayed in the right window; and in the animation, the thief will be hit and fall down after being hit.

An IDE user with IDE GUI open on the reaction edit model may want to add a new reaction just by traditional writing. To do this, the IDE user can click the icon "T" and type "The policeman

is pushing a pickup truck." The outcome may be that the input will be identified; and the demonstration animation of a model of policeman pushing a model of pickup truck will be displayed in the right window, and the truck will only be pushed forward very slowly if the strength related properties for the policeman is set higher than 50% of the value range, but if the strength related properties for the policeman is set very low, then he won't be able to push the truck forward.

An IDE user with IDE GUI open on the reaction edit model may want to save the reaction they added. To do this, the IDE user can click the "done" button, and the outcome may be that the reaction will be saved and updated on the reaction view.

An IDE user with IDE GUI open on the chaos screen may want to add a new card. To do this, the IDE user can click the icon called "add new branch" on toolbar, select a NPC, add new reactions, and click the "done" button. The outcome may be that the new card they added will be established.

An IDE user with IDE GUI open on the chaos screen may want to add a new exit. To do this, the IDE user can click the icon called "exit" and link an output of a card to this exit card. The outcome may be that the chaos and the story card will add a new exit.

An IDE user with IDE GUI open on the chaos screen may want to preview a chaos card or more chaos cards. To do this, the IDE user can select a card or several cards and click the icon call "preview" on toolbar. The outcome may be that the preview window will be shown to demonstrate the reactions.

An IDE user with IDE GUI open on the chaos screen may want to automatically generate more possibilities. To do this, the IDE user can click the icon called "generate," read it, and click the "green button" if it is reasonable. The outcome may be that new chaos structure will be generated.

An IDE user with IDE GUI open on the chaos screen may want to save a change and go to next step. To do this, the IDE user can click the first button on the toolbar to back to the story structure and click the "confirmed and build" button. The outcome may be that the change will be saved.

6. Chaos Box

6.1 Background

As discussed above, a chaos box (or chaos ball) can be the smallest logical unit in a specific movie / game scene. It defines the character in the scene and the behavior logic of the character in this scene.

In an example scenario of bank robbery, player Alex and several NPCs complete a story robbing a bank. The algorithm that implements the prediction model, including its technical architecture, data structure, and algorithm principles is described hereinafter.

6.2 Technical Structure

The core of the algorithm for model training is training data. In order to obtain a large amount of training data, the algorithm uses large-scale simulation to prepare data.

As illustrated in FIG. 22, after the attributes of the entity participating in the chaos ball have been defined, they are entered into the simulation engine 2202 for repeated simulation and

deduction. Through the rule 2201 constraint pruning 2208, a large amount of simulation process data 2203 can be obtained for the learning 2205 of the learning module 2209. A reward function 2204 can be used to provide feedback for the learning process, and the training strategy model is gradually improved as the amount of simulation data increases. The strategy model 2207 itself can also be provided back to the simulation process, improving training speed.

6.3 Simulation Engine

In a non-limiting example of the simulation engine, the progress of the scenario is divided into several steps. At each step, each entity (non-player character "NPC" or player) will have a current state, including personal attributes and relationships, from the perspective of the entity itself. When the simulation executes a certain step, the entity will respond to the received event (action). No action is also an action. Action may have an impact on the next step. This includes changing the status of the entity itself or others, or sending events to entities. These are predefined in the rules.

The simulation engine can accept a rule constraint, and the actions of all entities do not violate the rule. In all actions that do not violate the rules, the entity will obtain the action through its own strategy model. At the beginning of simulation, the strategy model is randomly selected. As the simulation progresses, the learning module will continuously update the strategy model.

After all entities in this step have completed the action, status change is completed according to the effect of the action, and whether the exit bar is reached will be checked. If the exit condition is reached, the simulation ends. Otherwise, the next step starts.

The exit rule is defined in the rule. When an entity's state reaches the exit rule, the simulation ends. When the number of steps reaches the set upper limit MAX_STEPS, the simulation also ends.

The status of each entity at each step of the simulation process, the events received, and the actions selected are recorded as training data for use by the learning module.

The player entity does not use the learned strategy model during simulation, but always randomly selects actions that conform to the rules.

6.4 Simplification

In theory, an entity can select an unlimited number of actions, but the algorithm needs to define all possible action sets in advance. In a restricted chaos ball scene, this requirement can usually be met.

In some embodiments, the simulation process can be simplified without affecting the algorithm verification. For example, the action decisions of all entities can be carried out uniformly only during step processing, without considering the situation where the decision frequencies of different entities are different. In some examples, only human entities are considered, while item entities are not considered for the time being.

In some examples, each entity generates only one action per step, and does not consider the case of multiple actions. Action itself can be some pre-defined compound actions, such as "drawing a pistol at the opponent and talking threatening to ask for money." In some examples, each entity only processes events sent to itself and does not observe events between others. If the

plot requires an action to have an effect on the onlookers, then this action can be implemented by explicitly generating an onlooker event to the onlookers. In some examples, for a speak event, NLP processing is not processed for the time being, and the intention is directed given directly in the event data. In some examples, no distinction between players and NPCs is made for the time being. In some other examples, relations are not implemented.

The implementation complexity of the algorithm can also be simplified by temporarily simulating all events with state changes. For example, the event that "I am targeted" can be expressed by the state that "a gun is pointing at me."

6.5 Simulation Rules

The simulation rules module can define a series of functions for use in the simulation process, including but not limited to the following examples:

Provide the current state of an entity and the events received and provide a list of optional actions;

Provide the current state of an entity, received events, and an action, and determine whether the action conforms to the rules;

Provide the current status of an entity, provide the end of the simulation if it has reached the exit condition, and if it ends, provide a description of the reason for the end;

Provide an entity and the action it sends, provide which entities will be affected by the action, including changing the state of the entity and sending events to the entity;

Provide an entity and its current state and provide its current benefits;

Provide a list of all entity state key values;

Provide a list of all event types; and

Provide a list of all action types.

6.6 Learning Module

The rule can define a reward function for each entity. When the exit condition is reached, each entity can calculate its revenue based on the current state as part of the training data. The learning module receives the states and actions in all steps of each simulation, and the entity's revenue when it finally exits, and uses this to train and update the strategy model of each entity for data training. Response action.

The strategy model trained by the learning module receives data such as entity, current state of the entity, received events, etc. as input, outputs a list of optional actions, and attaches a tendency information to each action. The tendency characterization model considers the contribution value of this action to the reward.

The specific algorithm used in this module will be described in more detail in the Algorithm Principle section.

6.7 Data Structure

The Simple type in this section refers to any type that can be Boolean, Integer, Float, or String. For the convenience of extension, except for those specifically pointed out, the compound data types in this section are all Dict types.

6.7.1 Entity

The entity represents an object in a chaos ball scene that can produce actions that affect the progress of the scene. The structure can be as follows:

Key value	Value type	Description
name	String	Entity name, unique within the scene
state	State	The state of the current entity

6.7.2 Entity State

The entity state represents a set of attributes that an entity has at a certain point in time that can affect the progress of the scene. It is a kv dictionary structure. The key is a String type, which will have different value sets depending on the scene and the complexity of the model. Different keys may have different value types; and in some embodiments, only Simple or Relation types are supported for simplicity. What kind of value a particular key corresponds to may stay the same during the entire simulation process.

Some examples of possible key / values:

- age: Integer
- frightened: the degree of fear of Float
- handing_gun: Boolean
- hands_positon: String of the position of the hand (raised above the head, desktop, reaching the alarm button, pressing the button, etc.)
- friendliness: Relation [Float]

6.7.3 Relationship Status

There is a special state of the entity, that is, the relationship with other entities. The particularity of the entity is that it has a more direct dimension, that is, the relationship to whom it is directed. Because there is a possibility that the two parties' perceptions of the relationship are inconsistent (for example, A likes B, but B does not like A), setting the relationship as an attribute of the entity may only represent the recognition of the relationship from the perspective of the entity to which it belongs. If it is an “objective” relationship, then it can be saved separately for both entities. When it needs to be modified, two events need to be sent to modify the status to synchronize the changes.

The data structure of the Relation is a kv dictionary, the key is the name of the target entity, and the value is the value recognized by the entity from the perspective of the relationship. The value is a Simple type.

6.7.4 State Change

State Change can represent the effect an action has on the state of an entity. The structure can be as follows:

Key value	Value type	Description
target	String	Target entity name

state	String	The state key to be changed; or "state key / target entity name" if it is a relationship state
relative	Boolean	Whether it is a relative change. If it is a relative change, the new state value is the old value + value; otherwise the new state value is set to value. The default is false. Note that relative can be true only when the status value type is Integer or Float
value	Simple	Value to increase (relative change) or value after change (absolute change)

6.7.5 Event

Events represent information sent to an entity that needs to be processed by the entity. It is a dictionary type and can have the following structure:

Key value	**Value type**	**Description**
type	String	Event type
sender	String	The name of the entity that caused the event
target	String	Name of the entity receiving the event
value	Any	Event content; the specific format is determined by the event type and can be empty

6.7.6 Action

Key value	**Value type**	**Description**
type	String	Action type
source	String	Name of the entity that initiated the action
value	Any	Action content; the specific format is determined by the action type and can be empty

6.7.7 Score

The return can be a Float with a value between -100 and 100. The closer to the expected result of the character, the more the value will approach 1.

6.7.8 Simulation Step (SimStepResult)

The simulation step describes the simulation result of a step in the simulation process. It is a Dict with the entity name as the key, and its value is the state of the entity, the events received, and the actions taken in this step. The structure can be as follows:

Key value	**Value type**	**Description**

state	State	The status at the beginning of the step (the status changes from the previous step have already taken effect)
events	List[Event]	All events received (i.e. all events for which the target is itself)
action	Action	The final action selected in this step
step	Integer	Step number (starting from 1)

6.7.9 Simulation Result (SimResult)

Simulation result can be a set of all steps from the beginning to the end of a simulation in a simulation. The structure can be as follows:

Key value	**Value type**	**Description**
steps	List[SimStepResult]	Simulation results at each step
exit_reason	String	Reason for the end of the simulation
scores	Dict[String,Score]	The revenue of each entity at the end of the simulation. The key value is the entity name.

6.7.10 Strategy

The optional action list returned by the strategy model can be a List with a structure of each element as follows:

Key value	**Value type**	**Description**
action	Action	Optional action
preference	Float	Value range: 0 to 1. The recommendation degree of the action that the model thinks

The Lists can be sorted by preference from high to low.

6.8 Algorithm Principle

The simulation and strategy part can adopt an algorithm model based on Reinforcement Learning (hereinafter referred to as RL) and gradually approximate the ideal strategy model through multiple iterations.

6.8.1 Basic Concepts of Reinforcement Learning

RL can use the method of learning while obtaining examples; after obtaining the examples, update its own model, use the current model to guide the next action, and update the model after the next action is rewarded, and iterate continuously; and repeat until the model converges. In this process, “how to choose the next action in the case of the current model is the best for improving the current model” is very important.

In RL, agents have clear goals. All agents can perceive their environment and guide their behavior according to goals. Therefore, another feature of RL is that it treats agents and the uncertain environment with which they interact as a complete question. In the RL problem, there are four very important concepts:

6.8.1.1 Rule (Policy)

Policy defines the behavior of agents in a specific environment at a specific time. It can be regarded as a mapping from environmental state to behavior, which is often expressed by π.

Policies can be divided into two categories:

Deterministic policy: $a = \pi(s)$

Stochastic policy: $\pi(a \mid s) = P[A_t = a \mid S_t = t]$,

wherein t is the time point, t = 0, 1, 2, 3, ...

$S_t \in S$, wherein S is a collection of environmental states, S_t is the state at time t, and S is one of the specific states;

$A_t \in A(S_t)$, $A(S_t)$ is a collection of actions in state S, A_t is the action at time t, and a is a specific action/behavior.

6.8.1.2 Reward Signal

The reward is a scalar value, which is the signal returned by the environment to the agent according to the behavior of the agent in each time step. The reward defines how well the behavior is performed in this scenario, and the agent can adjust its policy based on the reward. The reward is often represented by *R*.

6.8.1.3 Reward Function

The reward defines immediate returns, while a reward function defines long-term returns, which can be viewed as a cumulative reward, usually expressed by v.

6.8.1.4 A Model of The Environment

An example process of interaction between an agent and the environment can be represented by FIG. 23. In this figure, t is the time point, t = 0, 1, 2, 3... $S_t \in S$, wherein S is a collection of environmental states, $A_t \in A(S_t)$, $A(S_t)$ is a collection of actions in state S, and $R_t \in R \in \mathbb{R}$ (a numerical reward).

As illustrated in FIG. 23, at any time t, an agent (e.g., player or NPC) 2301 takes a corresponding action A_t 2302 according to its own state S_t 2303, where this action A_t 2302 is obtained from the simulation and/or reinforcement learning models on the backend. At the same time, the reward R_t 2304 can be obtained from the reward function. To any agent (e.g., player or NPC), the other agents are environmental variables 2305. The agent 2301 interacts with the environment variables 2305 according to its own state 2303, obtains the corresponding reward 2304, and updates the simulation model and reinforcement learning model according to the reward 2304.

6.8.2 Algorithm Steps

In the specific implementation of RL, the Q-Learnling algorithm can be used, which is essentially a Model-Free algorithm framework, without having to delve into the internal logic of the simulation environment to be implemented. In addition, the Q-Learning model also has the

characteristics of simple steps, good interpretability, and flexible algorithm framework, which is suitable as a basic framework for implementing prototypes. When dealing with complex environments, it can be upgraded to a DeepQ-Network (DQN) framework with a neural network.

The core of the Q-Learning algorithm is Q-Table, that is, a table is created for each agent, each row is state, each column is action, and the cell is the value corresponding to the corresponding state and action, that is, the Q value. The Q-Table of each agent is gradually approached through simulation. In the traditional Q-Learning model, it is generally a single-agent framework, and changes in the state of the agent are determined by changes in the surrounding environment of the agent. In a scenario that is a multi-agent mode, there can be multiple interactions between agents, and the actions or state changes of other agents can be used as the states that other agents can observe to build each agent's Q-Table. Initial Q-Table value can be 0:

	ACTION 1	**ACTION 2**	**ACTION 3**
STATE1	0	0	0
STATE2	0	0	0

The updated Q value can be obtained using the Bellmann Equation:

$$Q(s_t, a_t) \leftarrow (1-\alpha) \cdot \underbrace{Q(s_t, a_t)}_{\text{old value}} + \underbrace{\alpha}_{\text{learning rate}} \cdot \overbrace{\left(\underbrace{r_t}_{\text{reward}} + \underbrace{\gamma}_{\text{discount factor}} \cdot \underbrace{\max_a Q(s_{t+1}, a)}_{\text{estimate of optimal future value}} \right)}^{\text{learned value}}$$

Compared with the simple Greedy Search, this equation combines the forgetting factor and the learning rate, which can effectively avoid overfitting and improve the quality of learning.

The specific algorithm steps can be as follows

1. Initialize Q-Table matrix;
2. Each agent selects state;
3. According to Q-Table, select a possible action (a) under the current state (s);
4. Move to the next state (s');
5. Repeat step 3;
6. Update the Q-table using Bellman Equation;
7. Determine if the end condition is triggered: Yes, go to 8; No, go to 9;
8. Use the next state as the current state and return to step 3;
9. Determine whether the Q-Table of each agent is filled: Yes, go to 10; No, go to 2;
10. Simulation ends.

6.8.3 Verification Code

The verification code is in the hackaura repository, and it is implemented in Python 3.6+.

When a model is not specified, the simulation uses a random model to decide the actions of each entity, that is, randomly select one among all possible actions as the current step. After a few simulations, due to the randomness, the character often takes irrational actions. For example, in the bank robbery example, two characters may end up shooting each other.

The initial model trained using the score value obtained from the simulation as the algorithm input is located in the models directory. The -p parameter can be used to specify the model to use during simulation. Using the model simulation, the rally will lead to a decline in revenue, but after using the model, the chance of the shooting between the two characters can be greatly reduced. However, as some random factors may be still retained in the model, the shooting may still occur.

Due to the complexity of the scene, the current rules can still be further refined, and when the number of iterations of model training is not large enough, some unreasonable or irrational actions may still occur. With the improvement of the rules and continuous training, the situation can be gradually improved.

7. Hackaura Configuration File Format

7.1 Background

Hackaura is a simulation engine that assists in writing interactive movie or game scripts. The screenwriter sets up scenes through configuration files, characters, objects, moving rules, etc., and hackaura can simulate the scenes according to the configuration and the various possibilities of the story development.

The screenwriter writes scoring rules for the actions of the characters. Hackaura's MARL (Multi-Agent Reinforcement Learning) function can train models from the simulation results, allowing the characters to learn the training strategies that can obtain higher scores. As the number of simulation iterations increases, the model results will get better and better.

7.2 Configuration File

For ease of editing, an excel file (xlsx format) can be used as the configuration file. The program can convert it to JSON format before processing. Special configuration editing software can also be developed to directly edit the JSON file, and the configuration file in excel format can be discarded.

The excel file can be divided into multiple sheets, and each sheet is a table. Each table can have the following uniform formats:

- Use capital letters on the header;
- If the header of an example ends with "...," the data of this column needs to be combined with the right columns to form a list for processing.
- In the table, only the text content plays a role in scene simulation; and fonts, colors, and backgrounds do not affect program operation.
- All unique identifiers can be unique within the scene. For example, entities and objects are not uniquely identified the same.

7.3 Entities table

The entities table defines the entities in the scene, that is, the characters who have the ability to actively choose to move, generally the characters who participate in the scene. An entity table can have the following structure:

- ID column: the unique identifier of the entity, which must be a valid python variable name.
- NAME column: the display name of the entity.
- STATES ... column: the initialization status of the entity.

For example:

ID	NAME	STATES...		
alex	Alex	dead:bool=False	position:Position=LOBBY	fear:float=0

The above configuration defines an entity with the ID alex, the display name is Alex, and it has three status values:

- dead, boolean, initial value False, i.e., Alex is not dead.
- position, position type (defined in the StateEnums table), initial value LOBBY, i.e., Alex is in the hall.
- fear, floating-point number type, initial value 0, i.e., Alex's degree of fear is zero.

The types of state support can include bool, int integer type, float, enumeration types defined in the StateEnums table, etc. The values do not change during the static simulation. They may be only used for rules and can be ignored during training.

7.4 Objects Table

The objects table defines the objects in the scene. Objects are similar to entities, with unique logos, names, and states. However, the object may not actively select the movement, so unlike the entity, there is no definition of the movement rule for the object in the configuration file. The objects table may have the following structure:

- ID column: the unique identifier of the object, which can be a valid python variable name.
- NAME column: the display name of the object.
- STATES ... column: the initialization status of the object; the format can be the same as the Entities table.

7.5 StateEnums table

This table defines the enumeration types that will be used in the state and can have the following structure:

- NAME column: Enumerated type names, which are legal Python variable names.
- ITEMS ... column: all enumeration values of this enum type are legal python variable names (usually written in full characters).

For example:

NAME	ITEMS...			

Position	LOBBY	FRONT_DOOR	BACK_DOOR	OUT_OF_BANK

In this example, there are 4 enumeration values for the enumeration type of Position: LOBBY, FRONT_DOOR, BACK_DOOR, and OUT_OF_BANK. An entity can be in one of these 4 positions.

7.6 Actions Table

This table defines all actions that an entity can make.

Actions can have parameters indicating the entities or objects involved in action. For example, Shoot indicates that the automatic shooting and may require a target parameter to indicate to whom to shoot. There may be multiple parameters. For example, the action of ThrowBag, i.e., throwing a bag to a certain person may have two parameters bag and target, which indicate which bag to throw and to whom.

The action can have an implicit parameter source, which is the initiator of the action. This parameter does not need to be listed in the ID column and can be used directly.

The actions table can have the following structure:

- ID column: automatic unique identifier and parameter list. If there are no parameters, the column content is the automatic identifier, such as RaiseHands. If there are parameters automatically, they are separated by /, such as Shoot / target. When there are multiple parameters, the parameters are separated by, for example, ThrowBag / bag, target. Both the automatic identifier and parameter name can be valid python variable names.
- DESCRIPTION column: When the entity takes the action, the log content template output by the simulation engine. The parts enclosed in {and} in the template are replaced with the passed parameters during output. For example, the template is Point gun to {target} and Shoot. During the simulation, one character (e.g., Alex) points gun to the other character (e.g., Besty) and shoots. (Alex is source and Besty is target) will be the output.
- EFFECT column: When the entity makes this action, it will have an effect on the state of the entity or object. The contents of this column are in the form target.dead = True or target.fear + = 0.1. = Indicates that the specified state is set to the specified value. + = Means increase the specified state by the specified value (if the specified value is negative, the actual effect is to reduce the state value). + = may be used only for state types that can perform addition operations, such as int or float types.
- CONDITIONS… column: if there is a value, the listed EFFECT is effective only if all the listed expressions (because the table header is the end of…, thus a list) are all met; otherwise the EFFECT can be ignored. The expression content can be a python expression. The available variables include the automatic parameters, the unique identifier of the entity or object, the enumeration value defined in the StateEnums table, and random (for probability calculations).

- Each movement may have multiple effects, which may cause multiple state values to change. It is represented in the table by multiple characters, each of which is an effect. Starting from the second column, the NAME and DESCRIPTION columns can be left blank.

For example,

NAME	DESCRIPTION	EFFECT	CONDITIONS...
Shoot/target	points gun to{target}and shoots.	target.dead=True	
		target.point_gun_to=NONE	target.hasstate('point_gun_to')

The above rules indicate that if Alex does the Shoot / target move, when the target is Besty, the simulation engine will output Alex points gun to Besty and shoots. Then the dead state of Besty will be set to True. If the state set of Besty does not have point_gun_to, then the state will be set to the enumeration value of NONE.

7.7 Speak Table

This table defines the intention and corpus of Speak and can have the following structure:

- INTENT column: Intent name. Unique identifier, legal Python variable name.
- CORPORA... column: Corpus corresponding to intent. The first corpus will be displayed as the entity said during the simulation.

For example,

INTENT	CORPORA...
REQUEST_RAISE_HANDS	Put your hands up!

The above rules define a REQUEST_RAISE_HANDS intent (i.e., request raising hands), and provide a corpus Put your hands up !. During the simulation, if the entity executes Speak / target = Alex, intent = REQUEST_RAISE_HANDS, it will display the output Alex says: "Put your hands up!"

7.8 Exit table

This table defines the simulation exit conditions, that is, the simulation ends when any of the conditions defined in this table are satisfied. This table can have the following structure:

- REASON column: When this condition is met, the end reason of the output will be copied.
- CONDITIONS... column: When the listed expressions are all full, end the simulation. The format of the expression matches the CONDITIONS ... column in the Actions table.

If all the end conditions in the simulation process are not fully satisfied, the simulation will reach the maximum number of steps (specified when running) and will also exit.

For example,

REASON	CONDITIONS…	
Allpeopledied.	alex.dead	besty.dead

The above configuration indicates that when both Alex and Besty are dead, that is, when the dead state value is True, the simulation ends and the reason All people died is output.]

7.8 Rule- * Table

For each entity, there can be a table named Rule-ID (ID is the unique identifier of the entity), which defines what actions the entity can make under certain conditions. This table can have the following structure:

- ACTION column: The actions and parameters that the entity can initiate when the conditions of this CONDITIONS ... are satisfied. For example, Shoot / target = besty indicates that a response is initiated by a response. The target parameter is set to the entity corresponding to the unique identifier of besty.
- CONDITIONS ... column: When the listed expressions are all full, you can initiate a move. The format of the expression is the same as the CONDITIONS ... column in the Actions table, but the available variables are increased. The state of the entity can be used directly by the state name. It does not need to be used by the entity name +.

It should be noted that the rules limit what movements can be made after the full conditions are met. In one state, there may be many movements under the full conditions. What movements will the entity make during specific simulation can be determined by the trained model.

For example,

ACTION	CONDITIONS…		
Shoot/target=besty	notdead	has_gun	notbesty.dead

The above configuration indicates that when the entity is not dead and has a gun (has_gun), and Besty is alive (not besty.dead), the entity can make a shot at Besty (Shoot / target = besty) .

7.9 Score- * Table

For each entity, there can be a table named Score-ID (ID is the unique identifier of the entity), which defines the score of the entity at the end of the simulation, which is usually calculated from the state at the end of the simulation. This table can have the following structure:

- SCORE column: When the conditions of the CONDITIONS ... conditions are met, the score increases (the initial score is 0).
- CONDITIONS ... column: When the listed expressions are all full, the score is increased by the score given in the SCORE column. The expression format is consistent with the Rule- * table format.

The final score can be in the range between -100 as the lower limit and 100 as the upper limit.

For example,

SCORE	CONDITIONS…	
-100	dead	
100	notdead	got_money

The above configuration indicates that if the entity dies (dead), the score is -100; if it is not dead and gets money (got_money), the score is 100. Otherwise (survival but no money) the score is 0.

7.10 Character

The above mechanism can be used to configure personality. The following uses the character of compliance for Besty as an example to illustrate. By configuring Besty under the condition of low compliance and compliance, the simulation results will be different. The more compliance, the more inclined Besty will be to act according to others' requests.

Assuming that in the example storyline, Besty will be required to take two actions, raising hands and placing money. A line can be added to the StateEnums table to indicate these two requirements, and a NONE means that Alex is not required:

NAME	ITEMS…		
Order	NONE	RAISE_HANDS	PUT_MONEY

In the Entities table, the following states can be added for the Besty entity:

ID	NAME	STATE…		
Besty	Besty	obey:nfloat=0.3	fear:nfloat=0	ordered:Order=NONE

This table means that Besty's compliance is 0.3, and it is not afraid at first and is not required.

In the Actions table, certain actions can be configured to make demands on the opponent, such as some imperative words:

NAME	DESCRIPTION	EFFECT	CONDITIONS…
Speak/target,intent	says:"{intent}"	target.ordered=RAISE_HANDS	intent==REQUEST_RAISE_HANDS
		target.ordered=PUT_MONEY	intent==REQUEST_PUT_MONEY

Finally, the automatic rules in the Rule-Besty table of Besty can be configured. A probability model can be used to simulate whether Besty moves automatically according to requirements. That is, the probability of obey + fear will automatically move according to requirements. Otherwise, other eligible movements may be selected. This model makes Besty more inclined to move on demand when more afraid and only move on demand when obey + fear>= 1.

When editing the automatic rules, that is, for the required automatic conditions, no additional conditions need to be made (because unless other plot rules are not satisfied, the automatic condition is always in the optional set regardless of whether the probability is hit). For

other movements, conditions should be added so that this movement can be selected only when it is not required or the probability is missed (of course, other plot rules should also be met).

ACTION	CONDITIONS...		
NoAction	ordered!=RAISE_HANDS or random()>obey+fear	ordered!=PUT_MONEY or random()>obey+fear	
RaiseHands	ordered!=PUT_MONEY or random()>obey+fear	hands_position!=UP	
PutMoney/bag=money_bag	ordered!=RAISE_HANDS or random()>obey+fear	hands_position!= PUTTING_MONEY	money_bag.position== ACCOUNT_TABLE

In this way, in the simulation, when Besty receives a request, there will be a probability of fear + 0.3, and it is determined to act as requested. Since the action according to the request is also in the candidate pool for probability miss, the actual probability of movement according to the request is higher than fear + 0.3.

8. Training

8.1 Background

In an example bank robbery scene, there are 5 agents: two robbers Chance and Dozer; a bank teller Besty; a bank customer Doctor, whose occupation is a medical doctor; and a policeman Freeman, who may be an experienced policeman or a new policeman who just joined the force. The specific status is determined by the state of the story when Freeman appears.

The story is mainly divided into two parts. In the first part, Freeman does not appear, and the robbers can get $1,500. Freeman only appeared in the second part, and Dozer's goal is to get $150,000. It can usually take tens of time steps to complete a single episode. The time steps can be further refined, so the number of steps can increase further.

Each agent has their own executable action space. The current action space can be as follows:

- Chance: 62
- Dozer: 71
- Besty: 23
- Doctor: 21
- Freeman: 29

The action space can change with the development of the project. The overall trend is that each agent can perform more and more actions and become more flexible. What the algorithm engine does is to judge what actions each agent should perform except the player in the current state.

8.2 Models

Independent neural network models can be trained to predict the agents' respective action reward functions Q (s, a). An example training framework is DQN. On the basis of the original DQN, some validated tricks can be added, such as:

- DoubleDQN

- DuelingDQN
- Prioritized Replay
- Multi-step

Model exploration combines ε-exploration and RND strategies.

8.3 Alternate Training

These models can be trained alternately. The example pseudo code is as follows:

```
for outer_epoch in max_outer_epoch:
  for agent in agents:
    for episode in max_episode:
      Fix all the agents except agent;
      Simulate and collect simulation samples;
      Train agent model;
```

The advantage of this is that theoretically it can guarantee convergence. The main disadvantage is that the training speed can be very slow.

8.4 Simultaneous Training

The models can be trained simultaneously. The example pseudo code is as follows:

```
for episode in max_episode:
    Simulate and collect simulation samples;
    for agent in agents:
      Train agent by using samples from the newest simulations;
```

The process should be the same as the IQN in MARL, ignoring the possible interaction of multiple models.

The advantage is that the training speed becomes faster, but the disadvantage is that theoretically the model cannot be guaranteed to converge. However, from the specific training results, the final effect can still be acceptable.

Under this framework, the DQN can be changed to the A2C algorithm, and there may be no essential changes.

8.5 Integrated Training

To use multi-GPU resources to improve training efficiency, the agent models can be merged into one, calculating and training at the same time, and Loss can be trained to be the sum of all the Losses. FIG. 24 shows an example structure diagram of the integrated training.

While training using multi-GPU, the multi-process method can also be used to run new simulations to generate new sample data. The example pseudo code is as follows:

for id in range(num_procs):
Generate a new process to simulate and collect simulation samples; The new process updates its model periodically.

for episode in max_episode:
Train the whole model by using samples from the newest simulations.

Terminate all simulation processes.

8.6 Formulating Reward

The goal of the algorithm engine is to determine what actions each agent should perform when given the current state of the story. The so-called "reasonable" here refers to fitting human decision logic as much as possible, which is consistent with human nature. When people make decisions, they may not do so in full accordance with the maximum benefits, and they need to consider the human side. It is possible that in panic situations, human behavior may be irrational.

In some embodiments, the rewards of each agent can be considered to maximize revenue, without considering the human side. For every agent, the reward setting can mainly consider several aspects, such as

- Whether they were injured or died
- Whether they have achieved their goals, such as robbers grabbing enough money
- Are there other casualties?

8.7 Effectiveness of the Model

The effectiveness of each agent model can be determined by, for example, the following direct indicators:

Indicator 1: The current agent being tested uses the trained model to select the model from the candidate actions, while other agents use a random strategy to select the model from the candidate actions. In this case, the average cumulative gain obtained by the current agent is simulated multiple times. This cumulative gain can increase gradually as the model improves.

Indicator 2: At the beginning of the iteration, select some state action sets $\{(s_n, a_n)\}_{n=1}^{N}$ and calculate their Q values $\{Q(s_n, a_n)\}_{n=1}^{N}$. The Q value (distribution) can gradually increase as the model improves.

Indirect indicators of the model can also be used, for example:

- L2 loss of the model on the training set
- L2 loss of the model on the validation set
- L2 loss of the RND model on the training set
- L2 loss of RND model on validation set

Indirect indicators are generally relatively reasonable, and losses can be seen to decrease significantly. From the perspective of direct indicators, the indicators of many agent models may not be ideal. The fluctuations can be seen on the curve, but there may not be a significant increase.

FIGS. 25A and 25B show respective example cumulative income curves of Chance and Besty during a training session (30,000 episodes).

Generally, the Q value can be stable near a certain value after a certain episode, but there can be some cases that are not usual, as shown in FIG. 26A, which shows an example cumulative income curve of Besty during a training session (30,000 episodes). FIG. 26B shows an example cumulative income curve of Chance during the same training session (30,000 episodes).

Possible reasons that can lead to less than ideal direct indicators include but are not limited to the following:

- The benefit/revenue setting is unreasonable.
- In some states, the candidate action set is not set reasonably, or the set is too small.
- There are problems with the model training process.

8.7 Training and Simulation Efficiency

Simulation and model training can be performed using a multi-process method in a multi-CPU environment. The simulation can take hours, such as about 2 ~ 3 hours. It can simulate as many episodes as desired, for example, about 10,000 ~ about 1,000,000 episodes. The code can be optimized to speed up the simulation and training process in a multi-GPU environment. Rule constraints can also be added to the exploration space during training to reduce the invalid exploration space.

8.8 Hyperparameters

Hyperparameters can be divided into two broad categories:

8.8.1 Environmental hyperparameters

- Single step penalty step_cost: Using the trained model for simulation sometimes results in invalid repeated steps. A single step penalty can be to optimize this situation.
- One-step discount rate gamma (r): The larger r is, the more attention is paid to the final return.
- The maximum time step max_step allowed by an episode: During model training, if an episode is not completed within the max_step, the data generated by this episode is not added to the training data and can be directly discarded.

8.8.2 Model hyperparameters

- Replay pool size replay_buffer_size.
- Use RND to optimize the exploration efficiency: By using RND, the model can obtain similar results with less episode training data in the early stage; but after stabilization, the effect itself may not change significantly.
- Sample weight sample_weight: When using Prioritized Replay technology, some sample weight calculation methods can be used, and no significant difference was found compared to methods without using the sample weight.
- Input feature construction: The state values {st-3, st-2, st-1, st} of the last 4 time steps and their corresponding pairwise differences can be used to stitch together to

generate the input feature: [st-3 ; st-2; st-1; st; st-2-st-3; st-1-st-2; st-st-1], and the total dimension can be over about 4000.

- The corresponding hyperparameters of the model: The model can be a multi-layer FCN. The number of units in each layer, the overall number of layers, and so on can be all adjustable hyperparameters.

9. Model Visualization Scheme

9.1 Training process

The rewards curve of each agent can be visualized, as illustrated in FIG. 27.

9.2 Prediction Effect Of The Model At Different Stages

When used for action selection, the effect of the model at different stages (e.g., divided into multiple stages: beginning, during training, training completed) can also be visualized, as illustrated in FIG. 28. The bars indicate the predicted value of revenue obtained by the model for the corresponding action performed by an agent in the current state, and the bar lengths correspond to the revenue/gain/benefit value.

10. Cloud

The Chaos Box algorithm can be encapsulated on cloud, i.e., the backend 102, as described in great detail hereinafter. The core of the Chaos Box algorithm is the action prediction model for every character in the scenario, where the input is the status of the entire scenario and the action of the player as a character in it.

10.1 Entity

All the entities in the scenario can be defined, which include all the characters and objects. Every entity has its own attribute list, which will be set by the script writer and the algorithm team based on the actual character setting and the scenario. All the attributes can be divided into variable attributes (status values) and invariable attributes (attribute values). For instance, the former can be an attribute signifying a status such as “is_wounded” (the status value of being wounded or not), “fear” (the fear value) or the relationship between characters, and the latter can be a status signifying basic personality or ability such as “skill” (shooting skill value).

The attribute of objects can also be divided into status values and attribute values. For example, an invariable attribute of the gun as an object can be its “damage value”, and one of its variable attributes can be “bullet number” (the current number of the bullets of the owner or in the gun) which signifies a status.

10.2 Objects

All the interactive objects in the scene are in the list below, the number in front is the id of each object. Since different positions in the scene could be the target of animations, all the possible spots in the scene are listed, and ids are provided to them, making them a 'virtual' interactive objects.

Example entity objects are as follows:

0. env : virtual object represents the stage of the environment, could not be object or target of any animations, will be effected in the action_effect

1. bag : the bag Chance and Dozer bring to take the money

2. alarm : the silent alarm under the counter
3. safe : the safe
4. poorkey : key of Dozer's car
5. goodkey : key of McLaren GT which is the doctor's car
6. cashier : the cashier
7. safecodes : a small note which has the safecodes for the safe on it
8. twobags : the bags that Freeman will bring into the bank hall
9. phone : the cellphone that is in the bags Freeman will bring into the bank hall
10. gun_c : gun of Chance
11. gun_d : gun of Dozer
12. gun_f : gun of Freeman
13. picture : a picture of Chance's 3 month old baby
14. counter : bank counter
15. cash1 : the cash in the cashier
16. cash2 : the cash in the 8 two bags
17. cash3: the cash in the safe

Example position objects are as follows:

201. ENTRANCE : the entrance position of the bank hall
202. COUNTER : the position in front of the counter
203. CASHIER : the position in front of the cashier which is behind the counter
204. OUT : the position out of the bank hall
205. ALARM : the position in front of the alarm
206. SAFE : the position in front of the safe
207. CENTER: the center position of the bank hall
208. NOWHERE: the position of Freeman at game beginning (outside the view)

Position objects will be virtual and fixed, so they don't need to be included in the Session.

env

{"id":0,"name":"env","stage":"BEGIN"}

Possible values for **stage**: BEGIN / ANNOUNCED / REQUESTED / GOT_MONEY / KNOWN_NOT_ENOUGH / KNOWN_HOW_MUCH / REQUESTED_MORE

bag

{"id":1,"name":"bag","position":101,"money":0}

alarm

{"id":2,"name":"alarm","pressed":false}

safe

{"id":3,"name":"safe","open":true,"money":150000}

poorkey
{"id":4,"name":"poorkey","position":101}

goodkey
{"id":5,"name":"goodkey","position":103}

cashier
{"id":6,"name":"cashier","money":1500}

safecodes
{"id":7,"name":"safecodes","position":102}

twobags
{"id":8,"name":"twobags","position":104}

phone
{"id":9,"name":"phone","position":8}

gun1
{"id":10,"name":"gun1","position":100}

gun2
{"id":11,"name":"gun2","position":101}

gun3
{"id":12,"name":"gun3","position":102}

picture
{"id":13,"name":"picture","position":8}

counter
{"id":14,"name":"counter"}

cash1
{"id":15,"name":"cash1","position":6,"amount":1500}

cash2
{"id":16,"name":"cash2","position":8,"amount":150000}

10.3 Characters

The data structure will be updated during the iteration of the algorithm, every update will be informed through slack.

100. Chance : Robber A
101. Dozer : Robber B
102. Besty : the bank teller
103. Doctor : the customer who is also a doctor
104. Freeman : the policeman

10.3.1 Chance


```
{
    "id":100,
    "name":"Chanc
    e", "dead":
    false,
    "has_gun":
    true,
    "position": 201, // position will be the same as object, point to the id ofthe
  position object directly
    "protecting": null,        // protecting value will be the id of characters
    "point_gun_to": null,    // point_gun_to value will be the id of characters "pose":
    "STANDING",
    "trust_dozer": 1.0,
    "wounded": false,
    "order": null,
    "shown_picture": false,
    "waiting_for_police": false,
    "surrendered": false,
    "has_hit": true
}
```


10.3.2 Dozer


```
{
    "id":101,
```

```
        "name":"Dozer",
        "dead": false,
        "has_gun": true,
        "position": 201,
        "point_gun_to": null,
        "pose":
        "STANDING",
        "wounded": false,
        "has_shot": false,
        "order": null,
        "waiting_for_police": false,
        "surrendered": false,
        "has_hit": true
    }
```


10.3.3 Besty


```
    {
        "id":102,
        "name":"Besty"
        , "dead": false,
        "position": 203,
        "hands_position":
        "NORMAL", "pose":
        "STANDING",
        "wounded": false,
        "order": null,
        "opening_safe": false
    }
```


10.3.4 Doctor


```
    {
        "id":103,
        "name":"Doctor"
        , "dead": false,
        "has_gun": false,
        "position": 202,
```

"hands_position":
"NORMAL", "protecting":
null, "pose":
"STANDING",
"wounded": false,
"order": null,
"trust_chance": 0.0,
"helped": false
}

10.3.5 Freeman

{
"id": 104,
"name":
"Freeman",
"dead": false,
"has_gun": true,
"position": 204,
"first_year": true,
"announced_seniority": false
}

10.3.6 Position of each character

The position of the characters will only be referred to fixed position objects from 201-206. On the client side, this position will be set to the nearest position object of each character when every response action has been executed completely.

10.3.7 hands_position

NORMAL / UP / FACE

10.3.8 pose

STANDING / CROUCHED / LYING

10.3.9 order

null

COLLECT_MONEY / HOW_MUCH / SHE_NEEDS_HOSPITAL /
LET_HER_GO / NOBODY_CAN_LEAVE/BEG_FOR_LIFE /

MAKE_HIM_SHUT_UP / WHERE_IS_SAFE / DO_YOU_HAVE_KIDS / KEEP_DOCTOR_AS_HOSTAGE

10.4 Session

The set that consists of all the attributes of the characters and objects in the current scenario at a given moment is defined as a **session**, which signifies the status of the current scenario. Since the client obtains the actions to be implemented by every non-player character via HTTP requests, the contextual information of the current session is transmitted between the client and the cloud via the session.

The updating of the session is completed both on the cloud and the client. In case the user (player) makes a new action when the session has not changed (for example, when a character is implementing an action but has not completed it) and interrupts the actions of other characters, the new (combined) actions of the player will need to be re-uploaded based on the unchanged previous session, and other non-player characters will make new reactions and predictions based on the new action and return the list of actions to be implemented by every character.

Similarly, suspension and continuation of the game by the player can be logically controlled based on the session, and persistence of information of the objects in the scenario, such as their positions, status and angles of rotation, can be realized by the engine.

After the player implements the action "input," the following process occurs:

1. The player inputs an action, such as "speak" or an action input via the hand controllers;
2. The client transmits the action and current session to the cloud via HTTP request;
3. The cloud returns a list of reactions for every non-player character;
4. The client implements the actions for every non-player character;
5. When every action is successfully implemented, the client need to maintain the change of session with the action_effect data transmitted along with the reaction data after the action is successfully implemented (before the action is successfully implemented, the session data won't be changed). The local client will implement the corresponding object rule and sync the status of UI with the session;
6. Whenever the player inputs a new action, it will need to resend the action based on the current session. If the user interrupts an action that has not been completed by another character and input another action, it will send the new action to the cloud based on the current session; the cloud will return a new list of actions to be implemented by every character,and every non-player character will cease the current action and implement the new one;
7. Step 5 is repeated.

10.4.1 How will the client determine the action of a character is completed?

The specified sequence of actions is completed, and the physical reactions it triggers are completed. An action is considered completed when its subsequent physical reactions, such as that of beating, throwing or shooting, are completed.

10.4.2 A use case of an action being interrupted

The following is a specific example:

After the user implements action A, the client will send the session and action information to the cloud, which will generate the reaction for every characters. For example, if character A is to make action B (to press the alarm bell) but the user makes action C (gives a shout) when action B has not been completed (that is, character A is reaching the bell but has not pressed it), then the system will resend the current session and action C to the cloud (in theory, the current session will be the same as the previous one if no other action is implemented before action C after the user implements action A). The cloud will regenerate the reaction for every character (for example, reaction D for character A); after that reaction returns to the client, character A at the client will cease action B that is being implemented but not completed and implement action D instead.

For instance, if a customer is to press the alarm bell and shoot when he/she has not touched the bell, the reaction of the customer will be overwritten, and he/she may withdraw his hand, couch down and hold his/her head with arms. This is a specific case of interruption or so called reaction overwriting.

Sample session:

```
{
    "session": {
        "session_id": "43db98f6-8c9c-4cd6-b66e-
        c187341722bb", "chaos_box_id":1,
        "latest_screenshot":"http://xxx.png",
        "safe_code":"123456",
        "player_view_id":
        100, "current_timer":{
                "name":"ALARM_TIMER",
                "Length":60.0,      // decimal values
                "Rest_value":30.0
```

```
},
"objects": [
        {
            "id": 0,
            "name": "env",
            "stage": "BEGIN"
        },
        {
            "id": 1,
            "name": "bag", "position": 101,
            "money": 0

        },
```

```
{
    "id": 2,
    "name":
    "alarm",
    "pressed": false
},
{
    "id": 3,
    "name":
    "safe",
    "open": true,
    "money":
    150000
},
{
    "id": 4,
    "name":
    "poorkey",
    "position": 101
},
{
    "id": 5,
    "name":
    "goodkey",
    "position": 103
},
{
    "id": 6,
    "name":
    "cashier",
    "money": 1500
},
{
    "id": 7,
    "name":
    "safecodes",
    "position": 102
},
{
    "id": 8,
```

"name":
"twobags",
"position": 104
},
{
"id": 9,
"name":
"phone",
"position": 8
},
{
"id": 10,
"name":
"gun1",
"position":
100
},
{
"id": 11,
"name":
"gun2",
"position":
101
},
{
"id": 12,
"name":
"gun3",
"position":
102
},
{
"id": 13,
"name":
"picture",
"position": 8
},
{
"id": 14,

```
                "name": "counter"
            },
            {
                "id": 15,
                "name":
                "cash1",
                "position": 6,
                "amount": 1500
            },
            {
                "id": 16,
                "name":
                "cash1",
                "position": 8,
                "amount": 150000
            }
        ],
        "characters": [{
                "id": 100,
                "name": "Chance",
                "dead": false,
                "has_gun": true,
                "position": 201,
                "protecting": null,
                "point_gun_to": null,
                "pose":
                "STANDING",
                "trust_dozer": 1.0,
                "wounded": false,
                "order": null,
                "shown_picture": false,
                "waiting_for_police": false,
                "surrendered": false,
                "has_hit": true
            },
            {
                "id": 101,
                "name": "Dozer",
                "dead": false,
                "has_gun": true,
```

"position": 201,
"point_gun_to": null,
"pose":
"STANDING",
"wounded": false,
"has_shot": false,
"order": null,
"waiting_for_police": false,
"surrendered": false,
"has_hit": true
},
{
"id": 102,
"name":
"Besty",
"dead": false,
"position": 203,
"hands_position":
"NORMAL", "pose":
"STANDING",
"wounded": false,
"order": null,
"opening_safe": false
},
{
"id": 103,
"name":
"Doctor",
"dead": false,
"has_gun": false,
"position": 202,
"hands_position":
"NORMAL", "protecting":
null, "pose":
"STANDING",
"wounded": false,
"order": null,
"trust_chance": 0.0,
"helped": false
},

```
                {
                    "id": 104,
                    "name":
                    "Freeman",
                    "dead": false,
                    "has_gun": true,
                    "position": 204,
                    "first_year": true,
                    "announced_seniority": false
                }
            ],
            "extra_engine_data":{}
        }
    }
```


10.4.3 extra_engine_data

This data field is for the client to store the data needed to initialize the scene, including the coordinate of the root position of each characters and objects, the coordinate of the rigging joints of each characters, etc. When the players quit the application and enter the application again, they could resume their last session from this data field. This data field is decided by the client. When the session is initialized through API 2.2 request_new_session, this data field is empty, and client should add data into this data field in session. In the later communication between client and server, server won't make any changes to this data field.

10.4.4 safe_code

This data field represents the value of the safe code for the safe in the bank scene. This value will be different for different sessions, server will randomly generate value for this data field. The client should use this value to check whether player could open the safe or not, and also use this value to generate the content on the object 7 safe code small note.

10.4.5 current_timer

Timer is a mechanism for the client to implement 'count-down' features. In some cases, certain actions will trigger a count-down event in the scene such as pressing the alarm -> the police comes. The chaos box will need this environment status data to process for the output of different NPCs, so this information should be recorded in the session.

How timer will be triggered and how will the timer be maintained:

1. Certain actions will trigger the timer, so in the action_effect data field there will be expressions such as 'current_timer.name="ALARM_TIMER"/current_timer.length=60.0/current_timer.rest_value=60.0'. When this action has been completed, the information will be updated into session;
2. When the data has been updated into session, the client should know a timer called

'ALARM_TIMER' will be triggered right now with length 60 seconds. Every second the rest_value will be updated with minus 1.

3. When the rest_value turns to 0, call API 2.3 process_response, upload current session with current_timer.rest_value = 0 and empty input data field. Server will recognize the event as a timer set to 0 event, and process every NPCs response. rest_value = 0 is also one valid event that should be processed by each NPCs.

4. In the response of API 2.3, update the session in response to the local session, the current_timer will be set to empty (i.e. null).

10.4.6 extra_model_data

This data field is for the cloud model server to store the data needed to decide actions of NPCs. It is populated by server and opacity to the client.

10.5 Action

10.5.1 Overview

The definition of the actions. Unlike the definition of data structure required by the simulation system, the action definition here is the data structure defined for communication between the client and the cloud, which basically consists of the following fields:

```
{
    "id": 100,
    "name": "Chance", // who is executing the following animations
    "animations": [{
        "sequence": 1,
        "name": "aim",
        "target_id": 102,
        "object_id": 10,
        "context": {},
        "action_effect": []
    }, {
        "sequence": 1,
        "name": "speak",
        "target_id": 102,
        "object_id": -1,
        "context": {
            "content": "give me the money!"
        },
        "action_effect": []
    }] // client mainly need to take care of the animations part
}
```

The entity and object corresponding to the above **entity*id* / *object*id / target_id** could be extracted from the session of the current scenario. The action above is the sample data of an action of a non-player character, where the context fields of every animation object vary depending on different actions. For example, the context of the action "speak" is the **content** of what is said, and the action "point" has only an object while the context is null.

The context of all the actions is set as the list of the actions is set.

Every action animation has a specified sequence, that is, the order of implementations. The actions of the same sequence will be implemented at the same time. For different sequence numbers, actions of the next sequence can only be implemented after all actions of the previous sequence have been completed.

As described above, the entire action is considered completed only when the whole sequence of actions has been completed.

10.5.2 Animation Types

There are over 100 possible action types in the Chaos Box simulation system, but we can extract all the animations into following animation types including some certain context format. These following animation types are used for the communication between the server and the client.

In the sample given above, response action might consist of multiple animations as a sequence. action_effect will be explained later.

10.5.3 No Action

No certain action animation will be implemented.

```
{
    "sequence": 1,
    "name":
    "no_action",
    "target_id": -1,
    "object_id": -1,
    "context": {},
    "action_effect": []
}
```

10.5.4 Speak

One of the most common animations, and also need to call the request-voice API to extract certain voice resource to play.

```
{
```

```
        "sequence": 1,
        "name": "speak",
        "target_id": 102,
        "object_id": -1,
        "context": {
            "content": "give me the money!",
        },
        "action_effect": []
    }
```

10.5.5 Walk

Also one of the most common animations, since the context is complex, the client need to take care of different state of walk animation.

Normally the walk animation will only have target and empty object, the NPC will just walk to the target with the help of path finding system:

```
    {
        "sequence": 1,
        "name": "walk",
        "target_id": 103,
        "object_id": -1,
        "context": {
            "direction": "none",
            "object_state": "none"
        },
        "action_effect": []
    }
```

But still, in some circumstances, the walk animation will have object, it means i.e. walk in front of {object} towards {target}. So the direction in the context represents the position relationship of the NPC executing this animation to its object NPC. object_state is to specify the state of relationship between executing NPC and the object NPC, right now there are two possible states:

1. hostage: the executing NPC will hold the object NPC in his hand as a hostage and walk together;
2. guard: the executing NPC is facing the object NPC to protect target character behind.

```
    {
```

```
        "sequence": 1,
        "name": "walk",
        "target_id": 103,
        "object_id": 103,
        "context": {
            "direction": "behind/infront/beside/none",
            "object_state": "hostage/guard/none"
        },
        "action_effect": []
    }
```

10.5.6 Aim

```
    {
        "sequence":
        1, "name":
        "aim",
        "target_id": 103,       // use 103 Doctor as aiming target
        "object_id": 10,       // use gun as object
        "context": {},
        "action_effect": []
    }
```

If the target_id == -1, put gun down.

10.5.7 CouchDown

```
    {
        "sequence": 1,
        "name":
        "couchdown",
        "target_id": -1,
        "object_id": -1,
        "context": {},
        "action_effect": []
    }
```

10.5.8 Hop

```
{
        "sequence":
        1, "name":
        "hop",
        "target_id": 14,          // hop over the bank counter
        "object_id": -1,
        "context": {},
        "action_effect": []
}
```

10.5.9 Hand

```
{
        "sequence": 1,
        "name": "hand",
        "target_id": 102,
        "object_id": 1,        // hand 1 the bag to the target 102 Besty
        "context": {},
        "action_effect": []
}
```

10.5.10 Put

```
{
    "sequence": 1,
    "name": "put",
    "target_id": 1,
    "object_id": 15, // put cash1 into the target 1 bag
    "context": {
        "speed": "slow/fast"
    },
    "action_effect": []
}
```

10.5.11 Weight

```
{
```

```
        "sequence": 1,
        "name":
        "weight",
        "target_id": 1,          // weight the target 1 bag up and down by hand
        "object_id": -1,
        "context": {},
        "action_effect": []
}
```

10.5.12 Shoot

Shoot is also one of the most common animations. Main field of the context is the position to be shot. head/body means shooting in the target's head or body, if the target is not a character, then head or body will point to the same position. next_to means a missing shot, the bullet will fly pass the target and shoot the wall next to the target, making sure no other entities (objects/characters) will get shot.

```
{
    "sequence": 1,
    "name": "shoot",
    "target_id": 103,
    "object_id": 10,
    "context": {
        "position": "head/body/next_to"
    },
    "action_effect": []
}
```

10.5.13 Carry

```
{
        "sequence": 1,
        "name":
        "carry",
        "target_id": 204,          // carries object on his shoulder towards thetarget
        "object_id": 103,
        "context": {},
        "action_effect": []
}
```

10.5.13 PutHandsUp

```
{
"sequence": 1,
"name": "put_hands_up",
"target_id": 101,          // target could be the facing direction when he puts his
hands up
"object_id": -1,
"context": {},
"action_effect": []
}
```

10.5.14 Hit

```
{
"sequence": 1,
"name": "hit",
"target_id": 101,
"object_id": 10,          // hit {target} with {object}, if using the gun, it means
using the handle of the gun to hit the {target}
"context": {},
"action_effect": []
}
```

10.5.15 Press

```
{
"sequence": 1,
"name":
"press",
"target_id": 2,
"object_id": -1,
"context": {},
"action_effect": []
}
```

10.5.16 Run

```
{
        "sequence": 1,
        "name": "run",
        "target_id": 202,
        "object_id": -1,
        "context": {},
        "action_effect": ""
}
```

10.5.17 Throw

```
{
        "sequence": 1,
        "name":
        "throw",
        "target_id": 103,
        "object_id": 1,         // throw {object} over to {target}
        "context": {},
        "action_effect": ""
}
```

10.5.18 Wheel

```
{
        "sequence": 1,
        "name":
        "wheel",
        "target_id": 204,
        "object_id": 103,         // wheel is different from carry, carry is on the
 shoulder, wheel is more like drag the target character out
        "context": {},
        "action_effect": ""
}
```

10.5.19 Drive

Drive can be one of the ending animations of the scene, one of the endings is they walk out of the bank and drive their car away, then story ends. So drive animation is actually not driving the car but walk to the car and the view fade out.

10.5.20 Wait

```
{
    "sequence": 1,
    "name":
    "wait",
    "target_id": -1,
    "object_id": -1,
    "context": {},
    "action_effect": ""
}
```

10.5.21 ShakeHead

```
{
    "sequence": 1,
    "name": "shake_head",
    "target_id": 101,          // target could be the facing direction
    "object_id": -1,
    "context": {},
    "action_effect": ""
}
```

10.5.22 TakeOut

```
{
    "sequence": 1,
    "name":
    "take_out",
    "target_id": 8,
    "object_id": 9,        // take out {object} from {target}
    "context": {},
    "action_effect": ""
}
```

10.5.23 OpenSafe

```
{
    "sequence": 1,
    "name":
    "open_safe",
    "target_id": 3,
    "object_id": -1,
    "context": {},
    "action_effect": ""
}
```

10.5.24 WaveHands

```
{
    "sequence": 1,
    "name":
    "wave_hands",
    "target_id": 101,          // target could be the facing direction
    "object_id": -1,
    "context": {},
    "action_effect": ""
}
```

10.5.25 PullBehind

```
{
    "sequence": 1,
    "name":
    "pull_behind",
    "target_id": 101,
    "object_id": -1,          // pull {target} behind the executing NPC
    "context": {},
    "action_effect": ""
}
```

10.5.26 Disarm

```
{
```

```
        "sequence": 1,
        "name":
        "disarm",
        "target_id": 101,
        "object_id": -1,        // take {target}'s gun
        "context": {},
        "action_effect": ""
    }
```

10.5.27 NodHead

```
    {
        "sequence": 1,
        "name":
        "nod_head",
        "target_id": 101,           // target could be the facing direction
        "object_id": -1,
        "context": {},
        "action_effect": ""
    }
```

10.5.28 Slide

```
    {
        "sequence": 1,
        "name": "slide",
        "target_id": 101,
        "object_id": 1,         // put the {object} on the grounf and slide {object} over
    to {target}
        "context": {},
        "action_effect": ""
    }
```

10.5.29 Surrender

Surrender can also be one of the ending animations of the scene, one of the endings is Chance surrenders, then story ends. So surrender animation is actually the put_hands_up animation and a speak animation with intent 'SURRENDER', when the player plays as Chance and does such action, the story ends, when the player act as other NPCs and Chance as an NPC does such action, the story will also ends.

10.5.30 Take

```
{
    "sequence": 1,
    "name":
    "take",
    "target_id": 1,          // take {target} in his hands
    "object_id": -1,
    "context": {},
    "action_effect": ""
}
```

10.5.31 Scream

```
{
    "sequence": 1,
    "name":
    "scream",
    "target_id": 101,          // target could be the facing direction
    "object_id": -1,
    "context": {},
    "action_effect": ""
}
```

10.5.32 Handcuff

```
{
    "sequence": 1,
    "name":
    "handcuff",
    "target_id": 101,          // handcuffs {target}
    "object_id": -1,
    "context": {},
    "action_effect": ""
}
```

10.5.33 LookInto

```
{
    "sequence": 1,
    "name":
    "look_into",
    "target_id": 1,          // looks into {target}
    "object_id": -1,
    "context": {},
    "action_effect": ""
}
```

10.5.34 StartWalk

```
{
    "sequence": 1,
    "name": "startwalk",
    "target_id": 1,          // starts walking to {target}
    "object_id": -1,
    "context": {},
    "action_effect": ""
}
```

10.5.35 StandUp

```
{
    "sequence": 1,
    "name":
    "stand_up",
    "target_id": -1,
    "object_id": -1,
    "context": {},
    "action_effect": ""
}
```

10.5.36 Drop

```
{
```

```
            "sequence": 1,
            "name":
            "drop",
            "target_id": -1,
            "object_id": 10,       // drop {object}
            "context": {},
            "action_effect": ""
    }
```

10.6 action_effect

Once the animation has been successfully completed in the client, the client need to take care of the session update using the response data in the action_effect from every animations. We use our customized format of expression to represent the action_effect data.

There will be general format and specific format, in the general format, we use source or target to represent the affected entity:

```
source.{FIELD_NAME} = {FIELD_VALUE}
target.{FIELD_NAME} = {FIELD_VALUE}
```

For specific format, here are some samples:

```
Chance.trust_dozer -= 0.1
Doctor.trust_chance += 0.2
alarm.pressed = true
current_timer.name = “ALARM_TIMER”
```

Which basically could be represented as:

```
{ENTITY_NAME}.{FIELD_NAME} {OPERATOR} {FIELD_VALUE}
```

+=/-=/= operators are used to represent the change operation of values. Using entities' name to match the entity in the session, and update certain data field.

The action_effect data will be stored in a list of string value. Take “disarm” as an example:

```
 {
        "sequence": 1,
        "name":
        "disarm",
        "target_id": 101,
```

```
        "object_id": -1,        // take target's gun
        "context": {},
        "action_effect":["target.has_gun=false", "target.point_gun_to=null",
                         "source.has_gun=true"]
    }
```

10.7 Client

10.7.1 Observation

Normally, the client needs an observation system based on the perspective of every character, so that the characters will know if they need to make reactions to the actions of the player or other characters. But in this scenario of the bank we set now, all the characters other than the policeman who enters halfway (2 robbers, a bank teller and a customer) are in the scenario by default and can make reactions at any time based on their observation of each other's actions. Also, the policeman has a status attribute to indicate whether he is in the bank or not. The above observation does not need to be realized here as it has already been taken into account in the Chaos Box simulation and training.

10.7.2 Unique ID

Each client will have its own UUID during the first load of the first cinema scene. It will be used in all the following APIs.

10.8 Voice Recognition Streaming Service

This can be a socket connection between the client and the server based on socket.io.

When the volume of the microphone exceeds a certain level, the player is speaking something. The client will then upload the voice data with a fixed size package - 1000 bytes for now for one frame. Every frame contains a data field called voice_pid, packages with continuously same voice_pid make up the context for the complete voice data that we will process through our speech-to-text service or models.

When the microphone does not pick up any valid sound in a certain period of time (0.5s for now, could be changed), the client should send an event audio-end to the server. When audio-end has been sent, the voice_pid will be re-generated for the next possible voice package.

Example data structure for voice data:

```
{
  "UUID": "{client's uuid}", "voice_pid":
  "{voice package'suuid}", "timestamp":
  1234456778,
  "sample_rate_hertz": 16000,
  "voice_data": "Buffer",
}
```

When the cloud streaming service detect any speaking content, the server will send the data to the client through the socket. Then the client could consider the player has input a valid 'speak' action with the content.

```
{
  "voice_pid": "02928337",
  "speak_content": "give me the money",
}
```

10.8 API

FIG. 29 shows an example flow chart of the present invention. The API's shown in FIG. 29 will be described in detail below.

10.8.1 request_characters_available

This is the API to load the available characters for each client with UUID.

10.8.1.1 Address

/request_characters_available/.

10.8.1.2 Method

POST (Power-On Self-Test) can be used.

10.8.1.3 Data

UUID: unique ID of the requested client

10.8.1.4 Response

```
{
  "code": 1,
  "message":
  "", "data": {
        "UUID": "xxxx",
        "current_valid_session_id": "",
        "current_available_characters": [{
              "id": 100,
              "name": "Chance",
              "play_time": 1
        }, {
                    "id": 104,
                    "name": "Freeman",
                    "play_time": 0
        }],
```

10.8.2 request_new_session

This is the API for requesting a new session while initiating a new game.

For the session to start rolling, the first pack of response action list will be returned along with the newly created session data, the NPC will execute the initializing response action and then the story will start.

If a player does not do anything, then the player need to input their action respond to the first round of NPC actions. If the player hasn't input any valid input actions after 2 seconds after the completion of all the NPC's response actions, then no_action will be uploaded.

10.8.2.1 Address

/request_new_session/

10.8.2.2 Method

POST can be used.

10.8.2.3 Data

UUID: unique ID of the requested client

player_view_id: the id of the character that the client is going to use for initializing this new session.

10.8.2.4 Response


```
{
    "code": 1,
    "message":
    "", "data": {
        "UUID":"xxxx",
        "session": {
            "session_id":"43db98f6-8c9c-4cd6-b66e-c187341722bb",
            "chaos_box_id":1,
            "latest_screenshot":"",
            "safe_code":"123456", "player_view_id":
            100, "current_timer":{
                "name":"alarm",
                "length":60.0, "rest_value":30.0
            },
            "objects": [...],
            "characters": [...]
        },
        "response_actions": [{ "id": 101,
            "name": "Dozer",
            "animations": [{
                "sequence": 1, "name": "aim",
                "target_id": 102,
                "object_id": 10,
                "context": {}, "action_effect": []
        }, {
```

```
                                    "sequence": 1,
                                    "name": "speak",
                                    "target_id": 102,
                                    "object_id": -1,
                                    "context": {
                                        "content": "give me the money!"
                                    },
                                    "action_effect": []
                            }]
                    },{...}]
            }
    }
```

10.8.3 process_response

10.8.3.1 Address

/process_response/

10.8.3.2 Method

POST can be used.

10.8.3.3 Data

UUID: unique ID of the requested client; **session**: json string of full session data; **s**: json string of input actions.

```
{
        "UUID":"",
        "session":{
                    "session_id": "43db98f6-8c9c-4cd6-b66e-c187341722bb",
                    "player_view_id": 100,
                    "objects": [...],
                    "characters": [...]
        },
        "input":{
                    "name":"wave_hand",
                    "target_id": 102,
                    "object_id": -1,
                    "context":{},
        }
}
```

Here, input represents the input action from the player, including speaking action through the microphone and several actions from hand controllers (aim/wave_hand/nod_head/shake_head/hit/throw/handcuff….). The data structure is basically similar to the animation data structure, but without the action_effect data field, since the action triggered by the player doesn't need to take care of this part.

Client need to map every user input to a specific animation type, and upload the action input through this API.

10.8.3.4 Response

One special case: In certain circumstances, the **'dead'** property of the character Dozer will be changed after some specific action has been triggered by the player, this property will be updated by the server and returned in the **'session'** data field in the response of this API.


```
{
    "code": 1,
    "message": "",
    "data": {
        "UUID": "xxxx",
        "session": {
            "session_id": "43db98f6-8c9c-4cd6-b66e-
            c187341722bb",
            "chaos_box_id":1, "latest_screenshot":"http://xxx.png",
            "safe_code":"123456",
            "player_view_id": 100,
            "current_timer":{
                "name":"ALARM_TIMER
                ",
                "length":60.0,
                "Rest_value":30.0
            },
            "objects": [...],
            "characters": [...]
        },
        "response_actions": [{ "id": 101,
            "name": "Dozer",
            "animations": [{
                "sequence": 1, "name":
                "aim", "target_id": 102,
                "object_id": 10, "context":
                {}, "action_effect": []
            }, {
                "sequence": 1,
                "name": "speak",
                "target_id": 102,
                "object_id": -1,
                "context": {
```

```
                                        "content": "give me the money!"
                                    },
                                    "action_effect": []
                                }]
                    },{...}]
                }
        }
```

10.8.4 request_voice

This is the API for requesting voice data for certain dialogue content. Sound file can be in mp3 format. The response can be a link to the voice data on the AWS CDN.

10.8.4.1 Address

/request_voice/

10.8.4.2 Method

POST can be used.

10.8.4.3 Data

UUID: unique ID of the requested client

dialogue_content: string, content of the requested dialogue.

character_id: the id of the character who is going to speak this dialogue.

10.8.4.4 Response

```
{
    "code": 1,
    "message":
    "", "data": {
            "UUID": "xxxx",
            "dialogue_content": "give me the money!",
            "address": "https://xxx.mp3"
    }
}
```

10.8.5 request_session

This API can be used to get complete session data with single session id.

10.8.5.1 Address

/request_session/

10.8.5.2 Method

POST can be used.

10.8.5.3 Data

UUID: unique ID of the requested client

session_id: session id

10.8.5.4 Response

```
{
    "code": 1,
```

```
        "message": "",
        "data": {
                "UUID":"xxxx",
                "session": {
                        "session_id": "43db98f6-8c9c-4cd6-b66e-
                        c187341722bb", "chaos_box_id":1,
                        "latest_screenshot":"http://xxx.png",
                        "safe_code":"123456",
                        "player_view_id": 100,
                        "current_timer":{
                                "name":"alarm",
                                "length":60.0,
                                "Rest_value":30.0
                        },
                                "objects": [...],
                                "characters": [...]
                }
        }
}
```


10.8.6 upload_screenshot

This can be used to upload screenshot picture for specific session.

10.8.6.1 Address

/upload_screenshot/

10.8.6.2 Method

POST

10.8.6.3 Data

UUID: unique ID of the requested client

session_id: session id

img_content: image file content

10.8.6.4 Response


```
{
  "code": 1,
  "message":
  "", "data": {
        "UUID":"xxxx",
        "link":"http://xxxx.png"
  }
}
```

10.8.7 request_check_end

Every time session changed, this API can be called to check whether the current chaos box should be ended:

- Response action has been completed, action_effect has been updated on the session;
- Process_response receive updated session in the response from the server.

10.8.7.1 Address

/request_check_end/

10.8.7.2 Method

POST can be used.

10.8.7.3 Data

UUID: unique ID of the requested client

session: json string of full session data


```
{
    "UUID":"",
    "session": {
                "session_id": "43db98f6-8c9c-4cd6-b66e-
                c187341722bb", "chaos_box_id":1,
                "latest_screenshot":"http://xxx.png",
                "safe_code":"123456",
                "player_view_id":
                100,
                "current_timer":{
                        "name":"ALARM_TIMER",
                        "length":60.0,
                        "Rest_value":30.
                        0
                },
                "objects": [...],
                "characters": [...]
            }
}
```


10.8.7.4 Response


```
{
    "code": 1,
    "message":
    "", "data": {
            "UUID":"xxxx",
```

"session_id": "43db98f6-8c9c-4cd6-b66e-c187341722bb",
"is_end": false,
"next_chaox_box_id":1,
}
}

If is_end is true, it means it has reached the end of the current chaos box, and the id of next chaos box will be in next_chaos_box_id, this will be reflected in the data field chaos_box_id in session.

It should be understood that various changes and modifications to the example embodiments described herein will be apparent to those skilled in the art. Such changes and modifications may be made without departing from the spirit and scope of the present subject matter and without diminishing its intended advantages. It is therefore intended that such changes and modifications be covered by the appended claims.

Reference in the specification to, "examples," "various examples," "some examples," etc. means that a particular feature, structure, or characteristic described in connection with the example embodiments is included in at least one embodiment of the disclosure. The appearances of the above-referenced phrases in various places in the specification are not necessarily all referring to the same embodiment. Reference to embodiments is intended to disclose examples, rather than limit the claimed disclosure. While the disclosure has been particularly shown and described with reference to several embodiments, it will be understood by persons skilled in the relevant art that various changes in form and details may be made therein without departing from the spirit and scope of the disclosure.

It should be noted that the language used in the specification has been principally selected for readability and instructional purposes, and may not have been selected to delineate or circumscribe the inventive subject matter. Accordingly, the present disclosure is intended to be illustrative, but not limiting, of the scope of the disclosure.

It is to be understood that the figures and descriptions of example embodiments of the present disclosure have been simplified to illustrate elements that are relevant for a clear understanding of the present disclosure, while eliminating, for purposes of clarity, other elements, such as for example, details of system architecture. Those of ordinary skill in the art will recognize that these and other elements may be desirable for practice of various aspects of the present examples. However, because such elements are well known in the art, and because they do not facilitate a better understanding of the present disclosure, a discussion of such elements is not provided herein.

In some examples of the present methods and systems disclosed herein, a single component may be replaced by multiple components, and multiple components replaced by a single component, to perform a given command or commands. Except where such substitution would not be operative to practice the present methods and systems, such substitution is within the scope of the present disclosure. Examples presented herein, including operational examples, are intended

to illustrate potential implementations of the present method and system examples. Such examples are intended primarily for purposes of illustration. No particular aspect or aspects of the example method, product, computer-readable media, and/or system examples described herein are intended to limit the scope of the present disclosure.

The various components described herein may be and/or are executed by any suitable type of computing device including, for example, desktop computers, laptop computers, mobile phones, palmtop computers, personal data assistants (PDAs), etc. As used herein, a "computer," "computer system," "computing device," or "computing device," "machine," may be, for example and without limitation, either alone or in combination, a personal computer (PC), server-based computer, main frame, server, microcomputer, minicomputer, laptop, personal data assistant (PDA), cellular phone, pager, processor, including wireless and/or wireline varieties thereof, and/or any other computerized device capable of configuration for processing data for standalone application and/or over a networked medium or media. Computers and computer systems disclosed herein may include operatively associated memory for storing certain software applications used in obtaining, processing, storing, and/or communicating data. Such memory may be internal, external, remote, or local with respect to its operatively associated computer or computer system. Memory may also include any means for storing software or other instructions including, for example and without limitation, a hard disk, an optical disk, floppy disk, ROM (read-only memory), RAM (random-access memory), PROM (programmable ROM), EEPROM (extended erasable PROM), and/or other like computer-readable media.

The systems and devices described herein include various circuitries that may effectuate the present disclosure. The term circuitry used in the disclosure may include hardware components such as application-specific integrated circuits (ASICs), hardware interrupt controllers, hard drives, network adaptors, graphics processors, hardware based video/audio codecs, alone or in combination with the firmware/software used to operate such hardware. The term circuitry may also include microprocessors configured to perform function(s) by firmware, by switches, and/or one or more logical processors, e.g., one or more cores of a multi-core general processing unit. The logical processor(s) and the like may be configured by software instructions embodying logic operable to perform function(s) that are loaded from memory, e.g., RAM, ROM, firmware, etc. In example embodiments where circuitry includes a combination of hardware and software an implementer may write source code embodying logic that is subsequently compiled into machine executable code that may be executed by a logical processor, a microprocessor, or the like. one skilled in the art may appreciate that the state of the art has evolved to a point where there is little difference between hardware, software, or a combination of hardware/software, and the selection of hardware versus software to effectuate functions may merely a design choice. one of skill in the art may appreciate that a software process may be transformed into an equivalent hardware structure, and a hardware structure may itself be transformed into an equivalent software process.

Some portions of the above disclosure are presented in terms of methods and symbolic representations of operations on data bits within a computer memory. These descriptions and representations are the means used by those skilled in the art to most effectively convey the

substance of their work to others skilled in the art. A method is here, and generally, conceived to be a sequence of actions (instructions) leading to a desired result. The actions are those requiring physical manipulations of physical quantities. Usually, though not necessarily, these quantities take the form of electrical, magnetic, or optical signals capable of being stored, transferred, combined, compared, and otherwise manipulated. It is convenient at times, principally for reasons of common usage, to refer to these signals as bits, values, elements, symbols, characters, terms, numbers, or the like. Furthermore, it is also convenient at times, to refer to certain arrangements of actions requiring physical manipulations of physical quantities as modules or code devices, without loss of generality. It should be borne in mind, however, that all of these and similar terms are to be associated with the appropriate physical quantities and are merely convenient labels applied to these quantities. Unless specifically stated otherwise as apparent from the preceding discussion, throughout the description, discussions utilizing terms such as "processing" or "computing" or "calculating" or "determining" or "displaying" or "generating" or the like, refer to the action and processes of a computer system, or similar electronic computing device, that manipulates and transforms data represented as physical (electronic) quantities within the computer system memories or registers or other such information storage, transmission, or display devices.

Certain aspects of the present disclosure include process steps and instructions described herein in the form of a method. It should be noted that the process steps and instructions of the present disclosure may be embodied in software, firmware, or hardware, and when embodied in software, may be downloaded to reside on and be operated from different platforms used by a variety of operating systems.

The present disclosure also relates to an apparatus for performing the operations herein. This apparatus may be specially constructed for the required purposes, or it may comprise a general-purpose computer selectively activated or reconfigured by a computer program stored in the computer. Such a computer program may be stored in a computer-readable storage medium. The computer-eadable storage medium may be a tangible device that may retain and store instructions for use by an instruction execution device. The computer readable storage medium may be, for example, but is not limited to, an electronic storage device, a magnetic storage device, an optical storage device, an electromagnetic storage device, a semiconductor storage device, or any suitable combination of the foregoing. A non-exhaustive list of more specific examples of the computer readable storage medium includes the following: a portable computer diskette, a hard disk, a random access memory (RAM), a read-only memory (ROM), an erasable programmable read-only memory (EPROM or Flash memory), a static random access memory (SRAM), a portable compact disc read-only memory (CD-ROM), a digital versatile disk (DVD), a memory stick, a floppy disk, a mechanically encoded device such as punch-cards or raised structures in a groove having instructions recorded thereon, and any suitable combination of the foregoing, each coupled to a computer system bus. A computer readable storage medium, as used herein, is not to be construed as being transitory signals per se, such as radio waves or other freely propagating electromagnetic waves, electromagnetic waves propagating through a waveguide or other

transmission media (e.g., light pulses passing through a fiber-optic cable), or electrical signals transmitted through a wire. Furthermore, the computers and computer systems referred to in the specification may include a single processor or may be architectures employing multiple processor designs for increased computing capability.

Computer readable program instructions described herein may be downloaded to respective computing/processing devices from a computer readable storage medium or to an external computer or external storage device via a network, for example, the Internet, a local area network, a wide area network and/or a wireless network. The network may comprise copper transmission cables, optical transmission fibers, wireless transmission, routers, firewalls, switches, gateway computers and/or edge servers. A network adapter card or network interface in each computing/processing device receives computer readable program instructions from the network and forwards the computer readable program instructions for storage in a computer readable storage medium within the respective computing/processing device.

Computer readable program instructions for carrying out operations of the present disclosure may be assembler instructions, instruction-set-architecture (ISA) instructions, machine instructions, machine dependent instructions, microcode, firmware instructions, state-setting data, configuration data for integrated circuitry, or either source code or object code written in any combination of one or more programming languages. The computer readable program instructions may execute entirely on the user's computer, partly on the user's computer, as a stand-alone software package, partly on the user's computer and partly on a remote computer or entirely on the remote computer or server. In the latter scenario, the remote computer may be connected to the user's computer through any type of network, including a local area network (LAN) or a wide area network (WAN), or the connection may be made to an external computer (for example, through the Internet using an Internet Service Provider). In some embodiments, electronic circuitry including, for example, programmable logic circuitry, field-programmable gate arrays (FPGA), or programmable logic arrays (PLA) may execute the computer readable program instructions by utilizing state information of the computer readable program instructions to personalize the electronic circuitry, in order to perform aspects of the present disclosure.

Aspects of the present disclosure are described herein with reference to flowchart illustrations and/or block diagrams of methods, apparatus (systems), and computer program products according to embodiments of the disclosure. It will be understood that each block of the flowchart illustrations and/or block diagrams, and combinations of blocks in the flowchart illustrations and/or block diagrams, may be implemented by computer readable program instructions.

These computer readable program instructions may be provided to a processor of a general purpose computer, special purpose computer, or other programmable data processing apparatus to produce a machine, such that the instructions, which execute via the processor of the computer or other programmable data processing apparatus, create means for implementing the functions/acts specified in the flowchart and/or block diagram block or blocks. These computer readable program instructions may also be stored in a computer readable storage medium that may direct a computer,

a programmable data processing apparatus, and/or other devices to function in a particular manner, such that the computer readable storage medium having instructions stored therein comprises an article of manufacture including instructions which implement aspects of the function/act specified in the flowchart and/or block diagram block or blocks.

The computer readable program instructions may also be loaded onto a computer, other programmable data processing apparatus, or other device to cause a series of operational steps to be performed on the computer, other programmable apparatus or other device to produce a computer implemented process, such that the instructions which execute on the computer, other programmable apparatus, or other device implement the functions/acts specified in the flowchart and/or block diagram block or blocks.

The flowchart and block diagrams in the figures illustrate the architecture, functionality, and operation of possible implementations of systems, methods, and computer program products according to various embodiments of the present disclosure. In this regard, each block in the flowchart or block diagrams may represent a module, segment, or portion of instructions, which comprises one or more executable instructions for implementing the specified logical function(s). In some alternative implementations, the functions noted in the blocks may occur out of the order noted in the Figures. For example, two blocks shown in succession may, in fact, be executed substantially concurrently, or the blocks may sometimes be executed in the reverse order, depending upon the functionality involved. It will also be noted that each block of the block diagrams and/or flowchart illustration, and combinations of blocks in the block diagrams and/or flowchart illustration, may be implemented by special purpose hardware-based systems that perform the specified functions or acts or carry out combinations of special purpose hardware and computer instructions.

The methods and systems presented herein, unless indicated otherwise, are not inherently related to any particular computer or other apparatus. Various general-purpose systems may also be used with programs in accordance with the teachings herein, or it may prove convenient to construct more specialized apparatus to perform the disclosed method actions. The structure for a variety of these systems will appear from the above description. In addition, although some of the examples herein are presented in the context of a particular programming language, the present disclosure is not limited to any particular programming language. A variety of programming languages may be used to implement the teachings of the present disclosure as described herein, and any references above to specific languages are provided for disclosure of enablement and best mode of the present disclosure.

The term "computer-readable medium" as used herein may include, for example, magnetic and optical memory devices such as diskettes, compact discs of both read-only and writeable varieties, optical disk drives, and hard disk drives. A computer-readable medium may also include non-transitory memory storage that may be physical or virtual.

CLAIMS

What is claimed is:

1. A method comprising:

generating a chaos box for at least one of a scene, a character, an object, a story, a state, an action, or an event by a computing system comprising at least one processor and a data storage device in communication with the at least one processor, the chaos box comprising a basic unit comprising at least one variable of the at least one of a scene, a character, a story, a state, an action, or an event;

providing a first value to the at least one variable of the at least one of a scene, a character, an object, a story, a state, an action, or an event in the basic unit;

receiving an input comprising the at least one variable having a second value from an input device;

comparing the second value of the at least one variable of the input from the user with the first value of the at least one variable of the at least one of a scene, a character, an object, a story, a state, an action, or an event of the basic unit; and

providing an output based on the comparison.

2. The method of claim 1 comprising providing to the basic unit a filter selected from the group consisting of an event type filter, an event context filter, an observer self-status filter, a related non-player character (NPC) status filter, a relationship filter, and combinations thereof.

3. The method of claim 1 comprising generating the at least one of a scene, a character, an object, a story, a state, an action, or an event by a script analyzing unit of the computing system, the script analyzing unit comprising at least one of a character extraction unit, a scene parser, an entity extraction, or a modifier phrase parser unit.

4. The method of claim 1, wherein the input from the user is selected from the group consisting of a non-voice input, a voice input, and combinations thereof.

5. The method of claim 4, wherein the input from the user is a voice input, the method comprising converting the voice input to texts by an audio-to-text (ASR) unit of the computing system.

6. The method of claim 4, wherein the input from the user is a non-voice input, the method comprising receiving and/or analyzing the non-voice input by at least one of a controller, a sensor, and the computing system.

7. The method of claim 1 further comprising editing the at least one of a scene, a character,

an object, a story, a state, an action, or an event by changing the first value of the at least one variable of the at least one of a scene, a character, an object, a story, a state, an action, or an event.

8. The method of claim 7 further comprising generating an additional basic unit based on a pattern of the editing of the at least one of a scene, a character, an object, a story, a state, an action, or an event learned by the computing system.

9. A system comprising:

at least one processor;

a data storage device in communication with the at least one processor, wherein the data storage device comprises instructions that, when executed by the at least one processor, cause the at least one processor to

generate a chaos box for at least one of a scene, a character, an object, a story, a state, an action, or an event by a computing system comprising at least one processor and a data storage device in communication with the at least one processor, the chaos box comprising a basic unit comprising at least one variable of the at least one of a scene, a character, a story, a state, an action, or an event;

provide a first value to the at least one variable of the at least one of a scene, a character, an object, a story, a state, an action, or an event in the basic unit;

receive an input comprising the at least one variable having a second value from an input device;

compare the second value of the at least one variable of the input from the user with the first value of the at least one variable of the at least one of a scene, a character, an object, a story, a state, an action, or an event of the basic unit; and

provide an output based on the comparison.

10. The system of claim 9, wherein the data storage device comprises instructions that, when executed by the at least one processor, cause the at least one processor to provide to the basic unit a filter selected from the group consisting of an event type filter, an event context filter, an observer self-status filter, a related non-player character (NPC) status filter, a relationship filter, and combinations thereof.

11. The system of claim 9 comprising a script analyzing unit configured to generate the at least one of a scene, a character, an object, a story, a state, an action, or an event, the script analyzing unit comprising at least one of a character extraction unit, a scene parser, an entity extraction, or a modifier phrase parser unit.

12. The system of claim 9, wherein the input from the user is selected from the group consisting of a non-voice input, a voice input, and a combination thereof.

13. The system of claim 12, wherein the input from the user is a voice input, and the data storage device comprises additional instructions that, when executed by the at least one processor, cause the at least one processor to convert the voice input to texts by an audio-to-text (ASR) unit of the computing device.

14. The system of claim 12, wherein the input from the user is a non-voice input, and the data storage device comprises additional instructions that, when executed by the at least one processor, cause the at least one processor to receive and analyze the non-voice input by a physic engine.

15. The system of claim 9, wherein the data storage device further comprises instructions that, when executed by the at least one processor, cause the at least one processor to edit the at least one of a scene, a character, or a story by changing the first value of the at least one variable of the at least one of a scene, a character, or a story.

16. The system of claim 15, wherein the data storage device further comprises instructions that, when executed by the at least one processor, cause the at least one processor to generate an additional basic unit based on a pattern of the editing of the at least one of a scene, a character, or a story learned by the computing device.

17. A non-transitory computer-readable medium storing software comprising instructions executable by at least one processor which, upon such execution, cause the at least one processor to perform operations comprising:

generating a chaos box for at least one of a scene, a character, an object, a story, a state, an action, or an event by a computing system comprising at least one processor and a data storage device in communication with the at least one processor, the chaos box comprising a basic unit comprising at least one variable of the at least one of a scene, a character, a story, a state, an action, or an event;

providing a first value to the at least one variable of the at least one of a scene, a character, an object, a story, a state, an action, or an event in the basic unit;

receiving an input comprising the at least one variable having a second value from an input device;

comparing the second value of the at least one variable of the input from the user with the first value of the at least one variable of the at least one of a scene, a character, an object, a story, a state, an action, or an event of the basic unit; and

providing an output based on the comparison.

18. The non-transitory computer-readable medium storing software comprising instructions executable by at least one processor which, upon such execution, cause the at least one processor to perform operations further comprising editing the at least one of a scene, a character, or a story by changing the first value of the at least one variable of the at least one of a

scene, a character, or a story.

19. The non-transitory computer-readable medium storing software comprising instructions executable by at least one processor which, upon such execution, cause the at least one processor to perform operations further comprising generating an additional basic unit based on a pattern of the editing of the at least one of a scene, a character, or a story learned by the computing device.

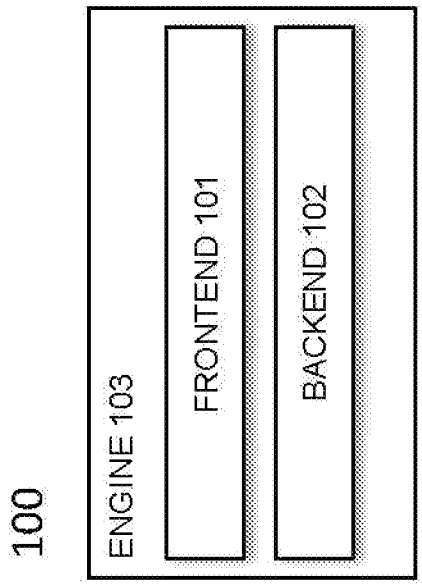


FIG. 1

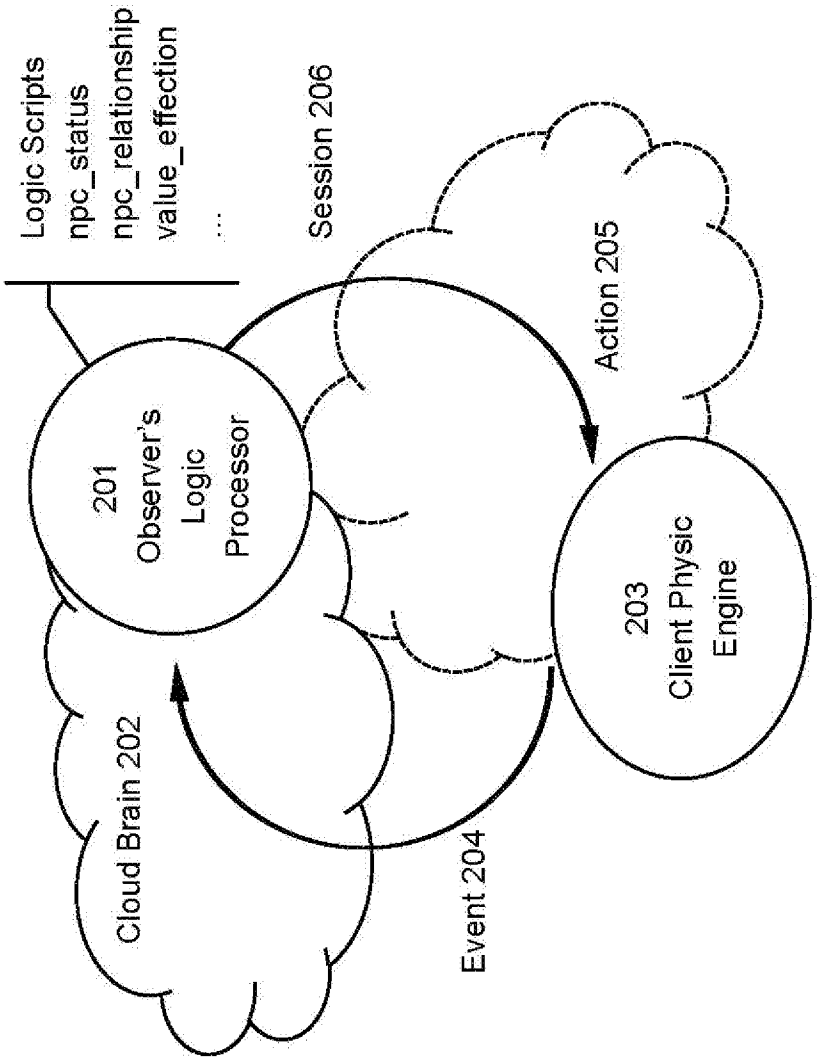

Logic Scripts
npc_status
npc_relationship
value_effection
....
Session 206
201
Observer's
Logic
Processor
Action 205
203
Client Physic
Engine
Cloud Brain 202
Event 204

FIG. 2

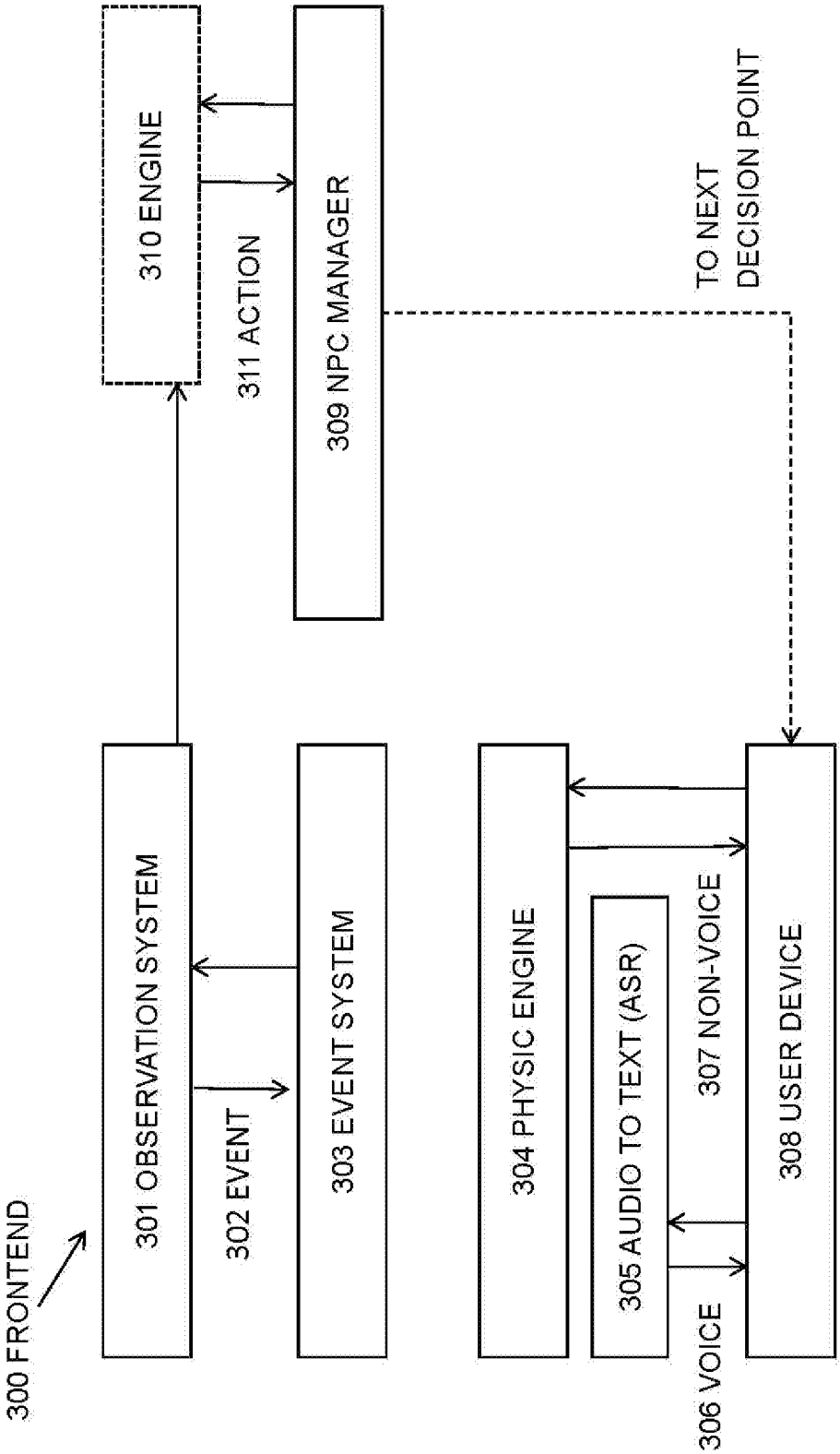

300 FRONTEND
301 OBSERVATION SYSTEM
302 EVENT
303 EVENT SYSTEM
304 PHYSIC ENGINE
305 AUDIO TO TEXT (ASR)
306 VOICE
307 NON-VOICE
308 USER DEVICE
309 NPC MANAGER
310 ENGINE
311 ACTION
TO NEXT DECISION POINT

FIG. 3

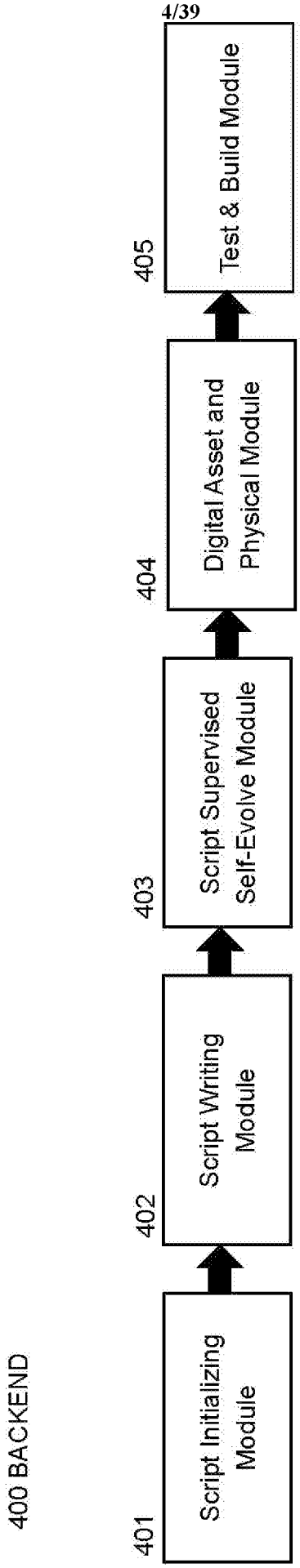

400 BACKEND
401
Script Initializing Module
402
Script Writing Module
403
Script Supervised Self-Evolve Module
404
Digital Asset and Physical Module
405
Test & Build Module

FIG. 4

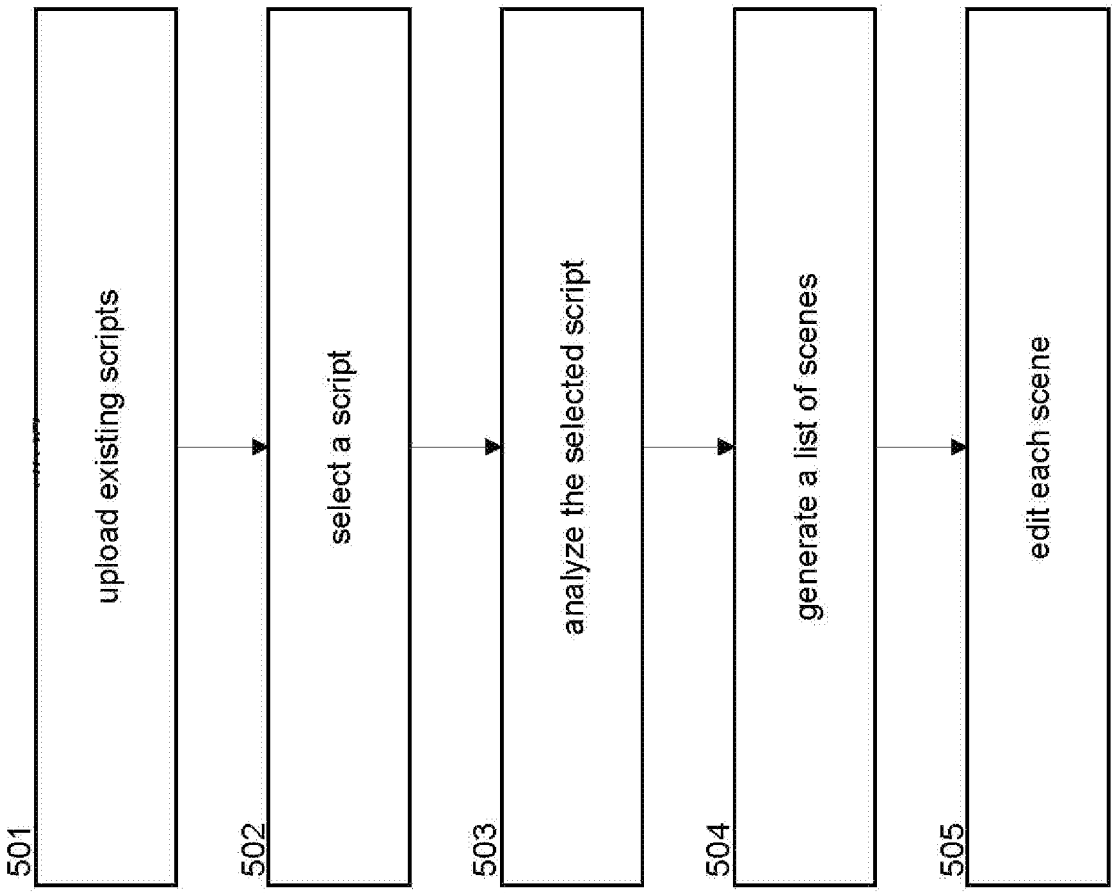

501
upload existing scripts
502
select a script
503
analyze the selected script
504
generate a list of scenes
505
edit each scene

FIG. 5

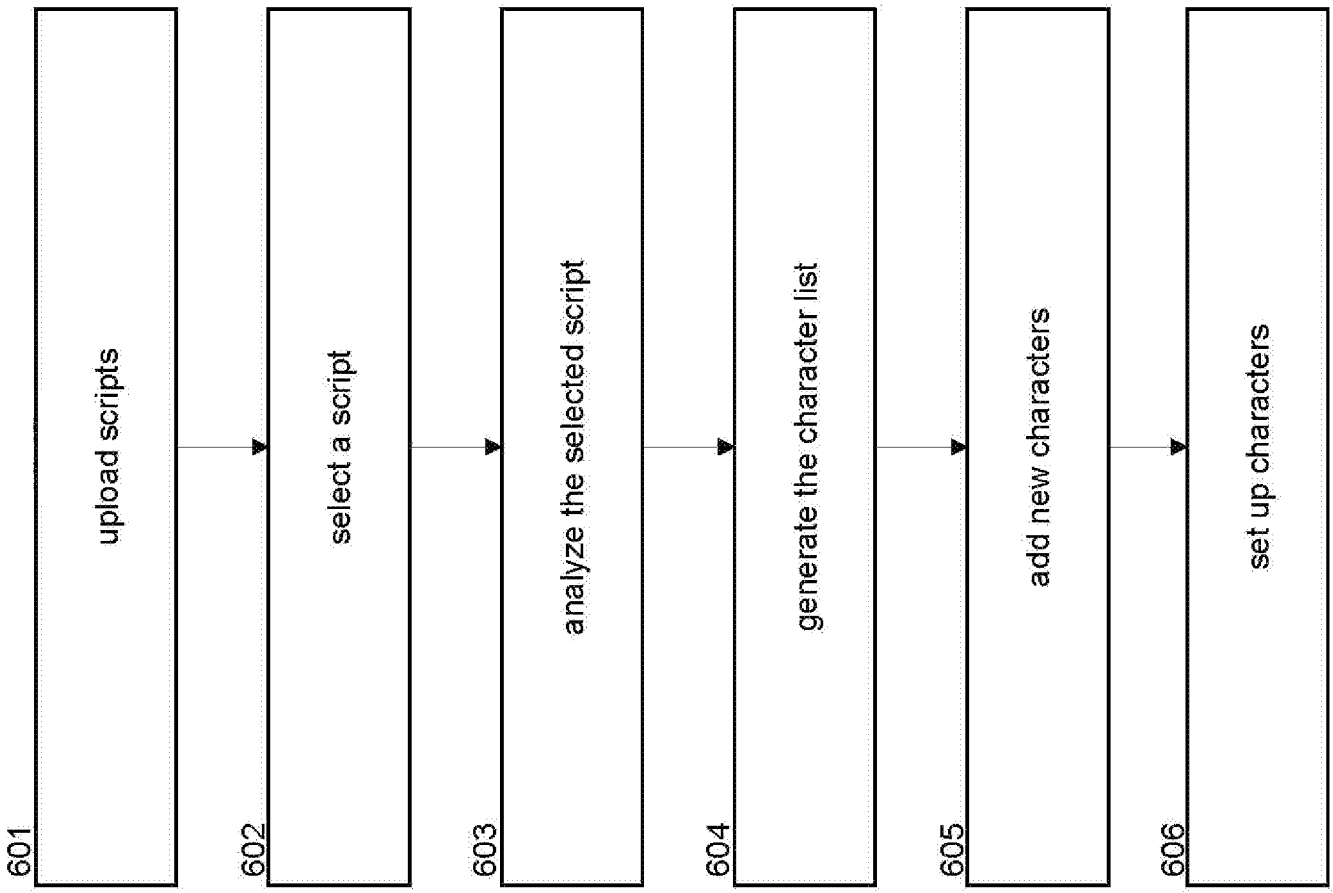

601
upload scripts
602
select a script
603
analyze the selected script
604
generate the character list
605
add new characters
606
set up characters

FIG. 6

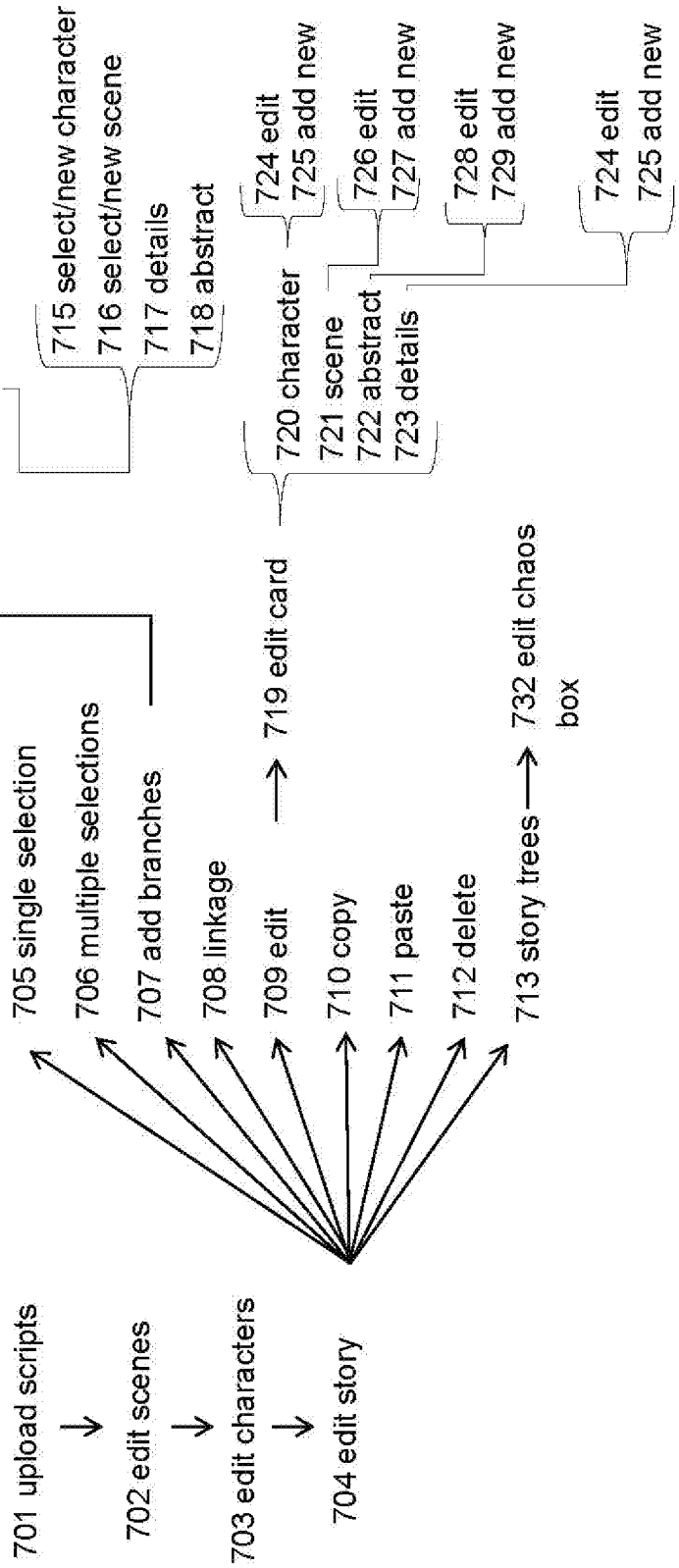

701 upload scripts
702 edit scenes
703 edit characters
704 edit story
705 single selection
706 multiple selections
707 add branches
708 linkage
709 edit
710 copy
711 paste
712 delete
713 story trees
714 blank card
715 select/new character
716 select/new scene
717 details
718 abstract
719 edit card
720 character
721 scene
722 abstract
723 details
724 edit
725 add new
726 edit
727 add new
728 edit
729 add new
732 edit chaos box
724 edit
725 add new

FIG. 7

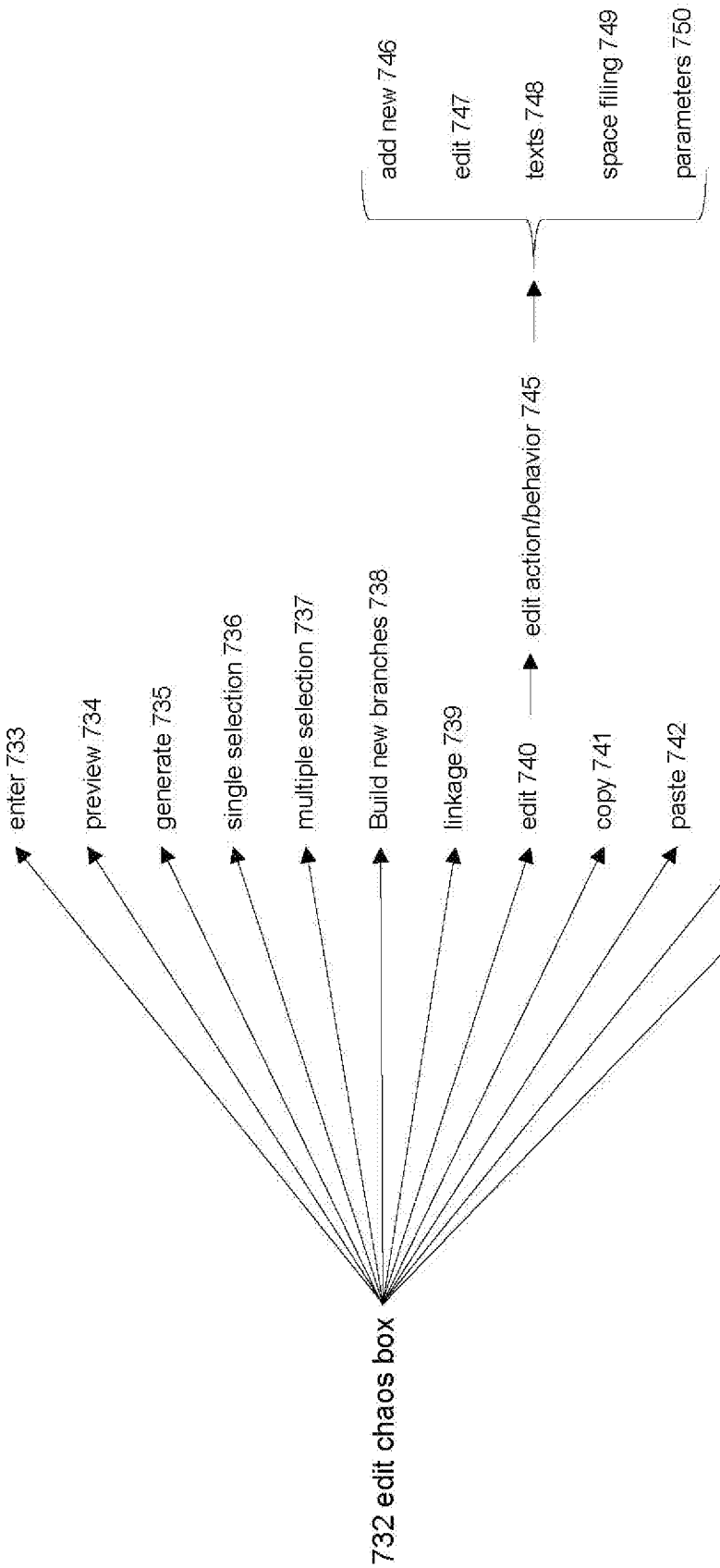

732 edit chaos box
enter 733
preview 734
generate 735
single selection 736
multiple selection 737
Build new branches 738
linkage 739
edit 740
copy 741
paste 742
delete 743
card 744
edit action/behavior 745
add new 746
edit 747
texts 748
space filing 749
parameters 750

FIG. 8

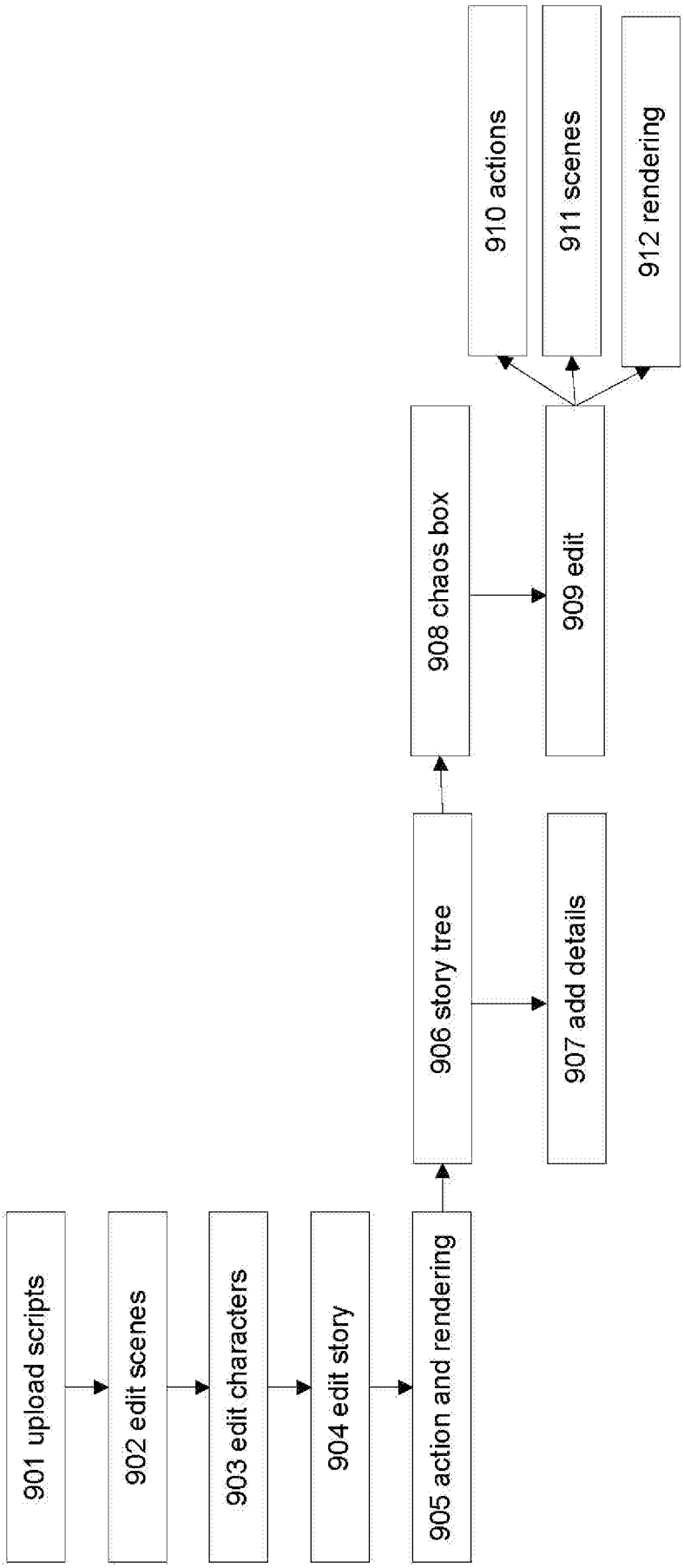

901 upload scripts
902 edit scenes
903 edit characters
904 edit story
905 action and rendering
906 story tree
907 add details
908 chaos box
909 edit
910 actions
911 scenes
912 rendering

FIG. 9

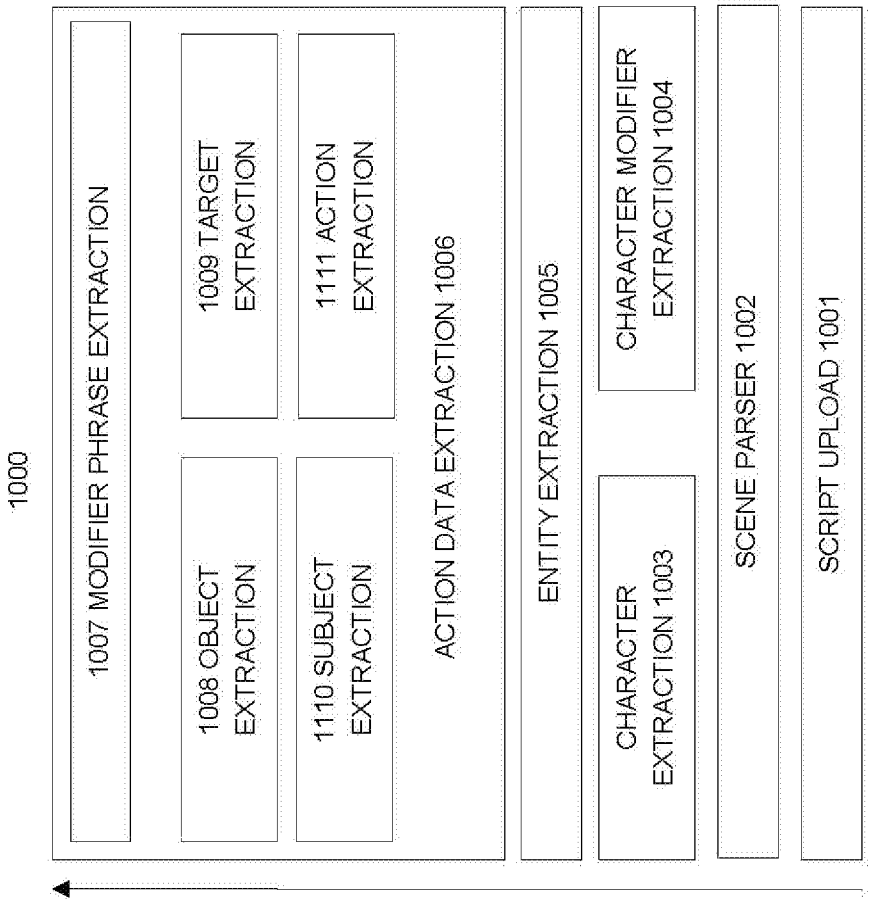

1000
1007 MODIFIER PHRASE EXTRACTION
1008 OBJECT EXTRACTION
1009 TARGET EXTRACTION
1110 SUBJECT EXTRACTION
1111 ACTION EXTRACTION
ACTION DATA EXTRACTION 1006
ENTITY EXTRACTION 1005
CHARACTER EXTRACTION 1003
CHARACTER MODIFIER EXTRACTION 1004
SCENE PARSER 1002
SCRIPT UPLOAD 1001

FIG. 10

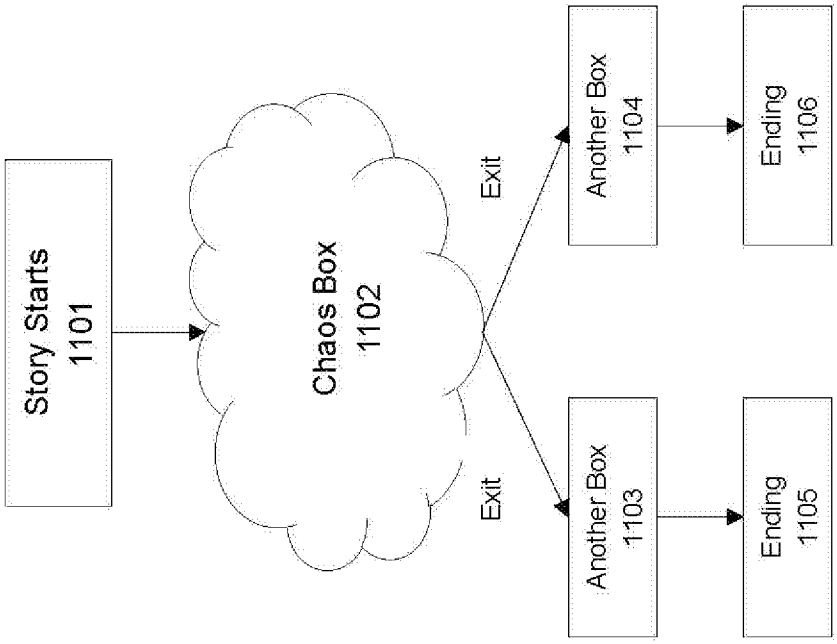

Story Starts
1101
Chaos Box
1102
Exit
Exit
Another Box
1104
Another Box
1103
Ending
1106
Ending
1105

FIG. 11

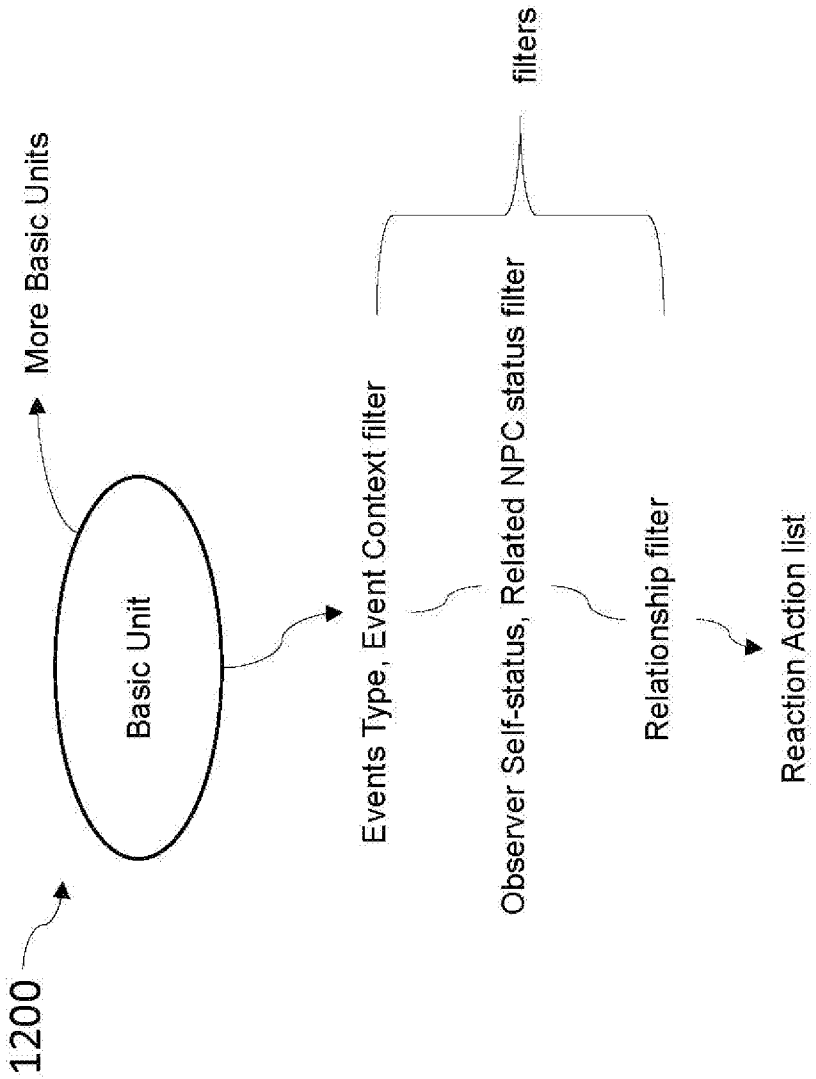

1200
Basic Unit
More Basic Units
Events Type, Event Context filter
Observer Self-status, Related NPC status filter
Relationship filter
filters
Reaction Action list

FIG. 12

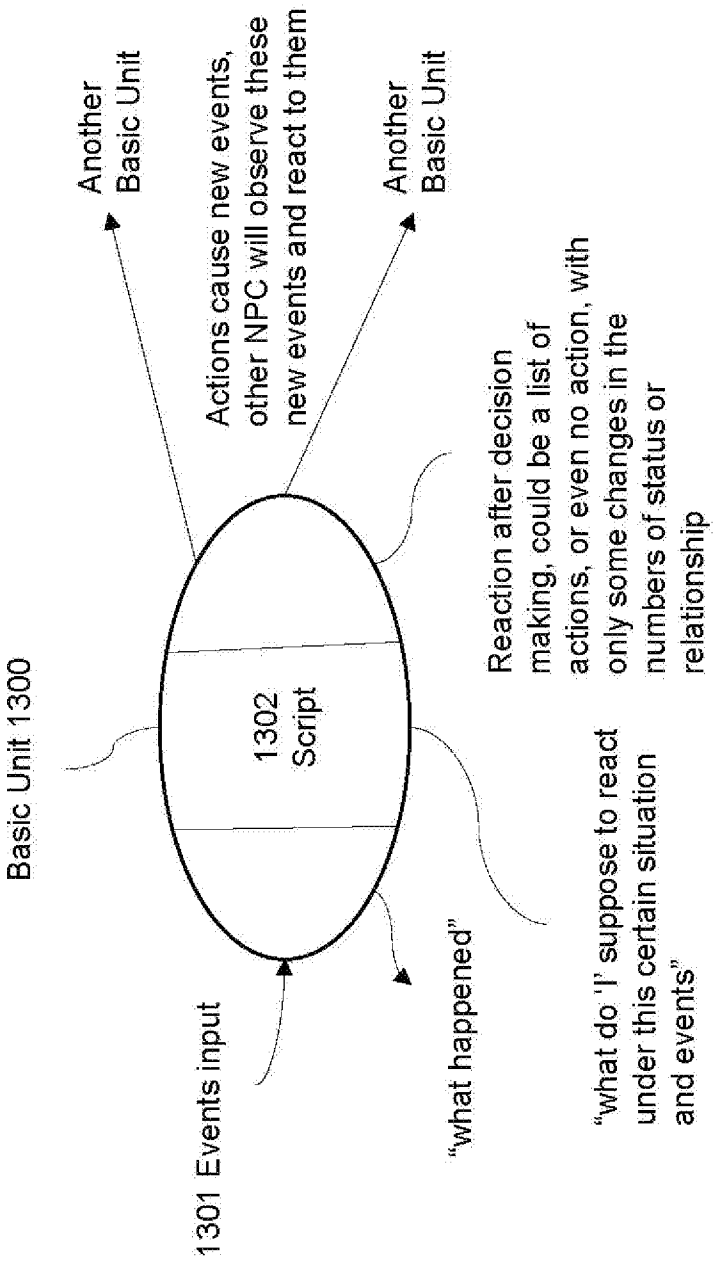

Basic Unit 1300
1301 Events input
1302
Script
"what happened"
"what do 'I' suppose to react under this certain situation and events"
Reaction after decision making, could be a list of actions, or even no action, with only some changes in the numbers of status or relationship
Actions cause new events, other NPC will observe these new events and react to them
Another Basic Unit
Another Basic Unit

FIG. 13

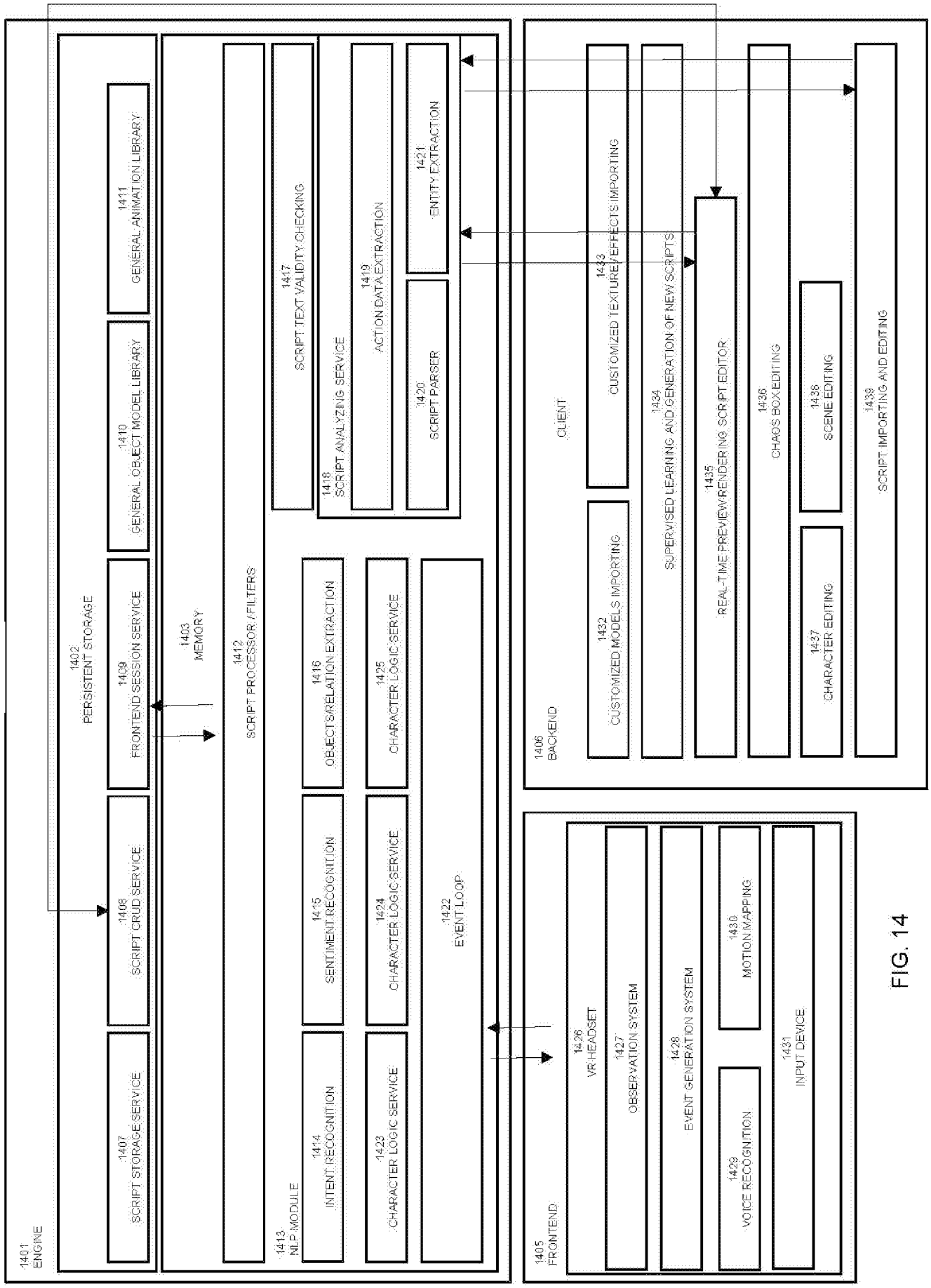

1400
1401 ENGINE
1402 PERSISTENT STORAGE
1407 SCRIPT STORAGE SERVICE
1408 SCRIPT CRUD SERVICE
1409 FRONTEND SESSION SERVICE
1410 GENERAL OBJECT MODEL LIBRARY
1411 GENERAL ANIMATION LIBRARY
1403 MEMORY
1412 SCRIPT PROCESSOR / FILTERS
1413 NLP MODULE
1414 INTENT RECOGNITION
1415 SENTIMENT RECOGNITION
1416 OBJECTS/RELATION EXTRACTION
1417 SCRIPT TEXT VALIDITY CHECKING
1418 SCRIPT ANALYZING SERVICE
1419 ACTION DATA EXTRACTION
1420 SCRIPT PARSER
1421 ENTITY EXTRACTION
1423 CHARACTER LOGIC SERVICE
1424 CHARACTER LOGIC SERVICE
1425 CHARACTER LOGIC SERVICE
1422 EVENT LOOP
1405 FRONTEND
1426 VR HEADSET
1427 OBSERVATION SYSTEM
1428 EVENT GENERATION SYSTEM
1429 VOICE RECOGNITION
1430 MOTION MAPPING
1431 INPUT DEVICE
1406 BACKEND
CLIENT
1432 CUSTOMIZED MODELS IMPORTING
1433 CUSTOMIZED TEXTURE/EFFECTS IMPORTING
1434 SUPERVISED LEARNING AND GENERATION OF NEW SCRIPTS
1435 REAL-TIME PREVIEW/RENDERING SCRIPT EDITOR
1436 CHAOS BOX EDITING
1437 CHARACTER EDITING
1438 SCENE EDITING
1439 SCRIPT IMPORTING AND EDITING

FIG. 14

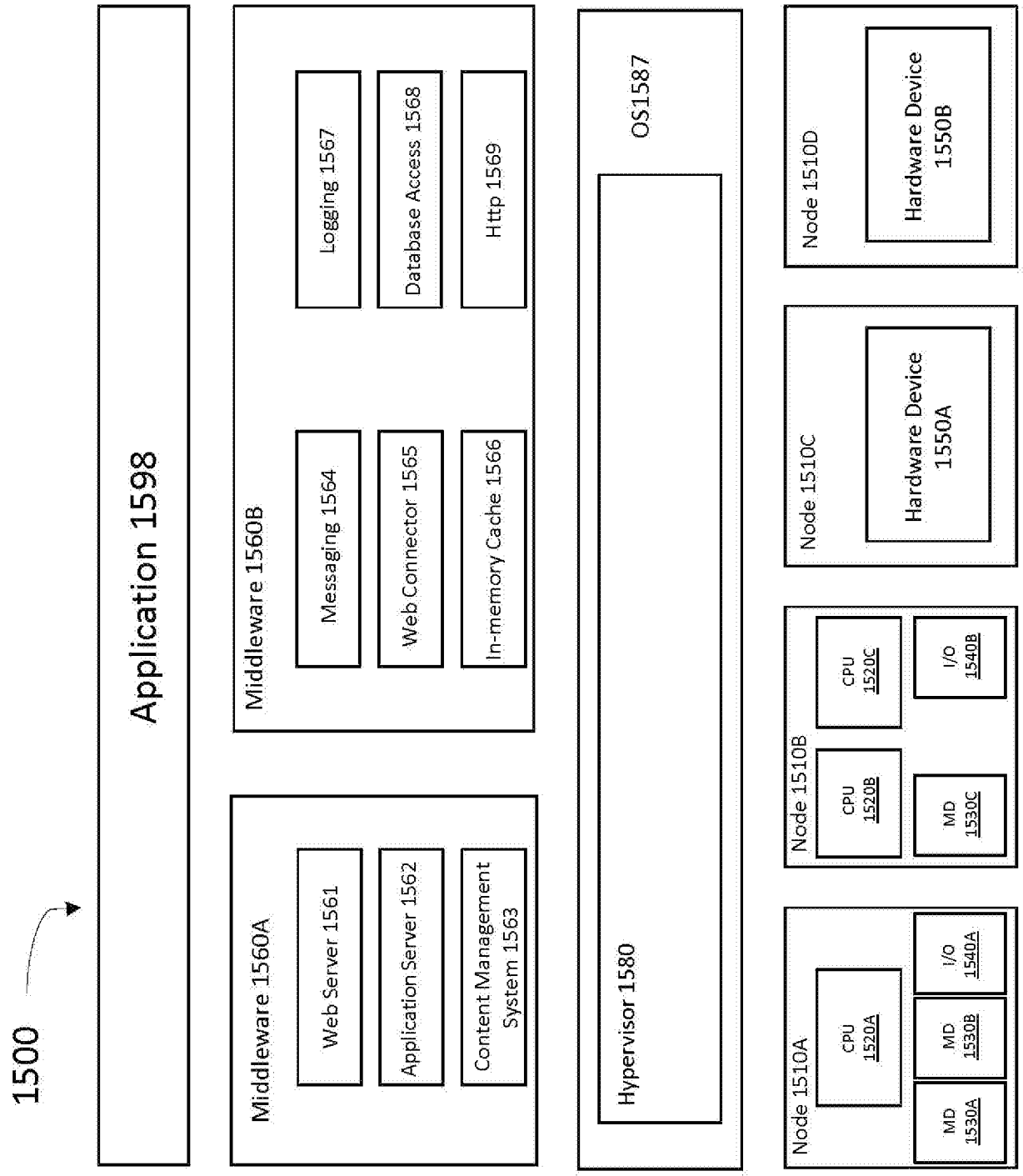

1500
Application 1598
Middleware 1560A
Web Server 1561
Application Server 1562
Content Management System 1563
Middleware 1560B
Messaging 1564
Web Connector 1565
In-memory Cache 1566
Logging 1567
Database Access 1568
Http 1569
Hypervisor 1580
OS1587
Node 1510A
CPU 1520A
MD 1530A
MD 1530B
I/O 1540A
Node 1510B
CPU 1520B
MD 1530C
CPU 1520C
I/O 1540B
Node 1510C
Hardware Device 1550A
Node 1510D
Hardware Device 1550B

FIG. 15

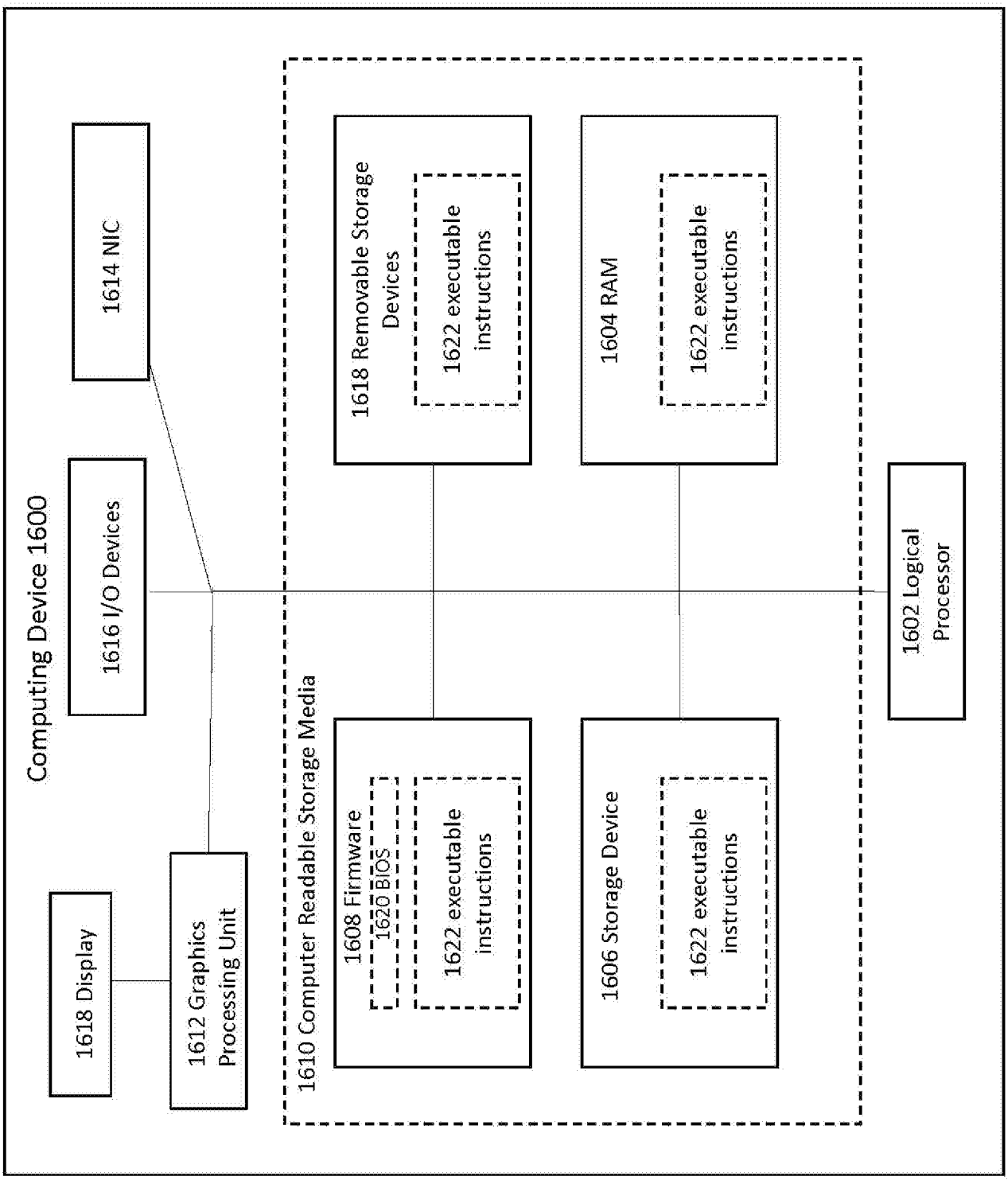

Computing Device 1600
1614 NIC
1616 I/O Devices
1618 Display
1612 Graphics Processing Unit
1610 Computer Readable Storage Media
1618 Removable Storage Devices
1622 executable instructions
1604 RAM
1622 executable instructions
1608 Firmware
1620 BIOS
1622 executable instructions
1606 Storage Device
1622 executable instructions
1602 Logical Processor

FIG. 16

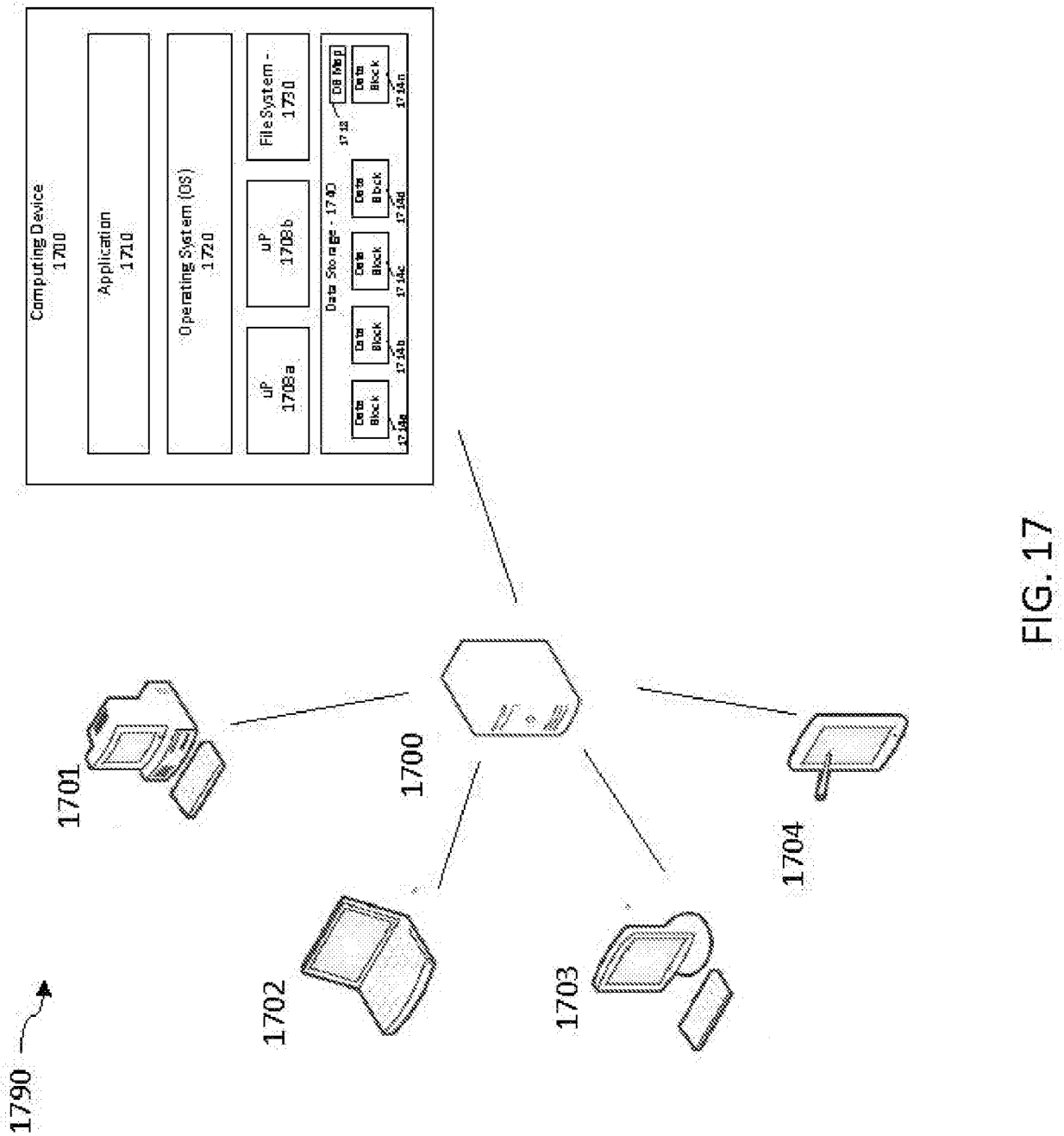

Computing Device
1700
Application
1710
Operating System (OS)
1720
uP
1708a
uP
1708b
FileSystem -
1730
Data Storage - 1740
Data Block
DB Map
1790
1700
1701
1702
1703
1704

FIG. 17

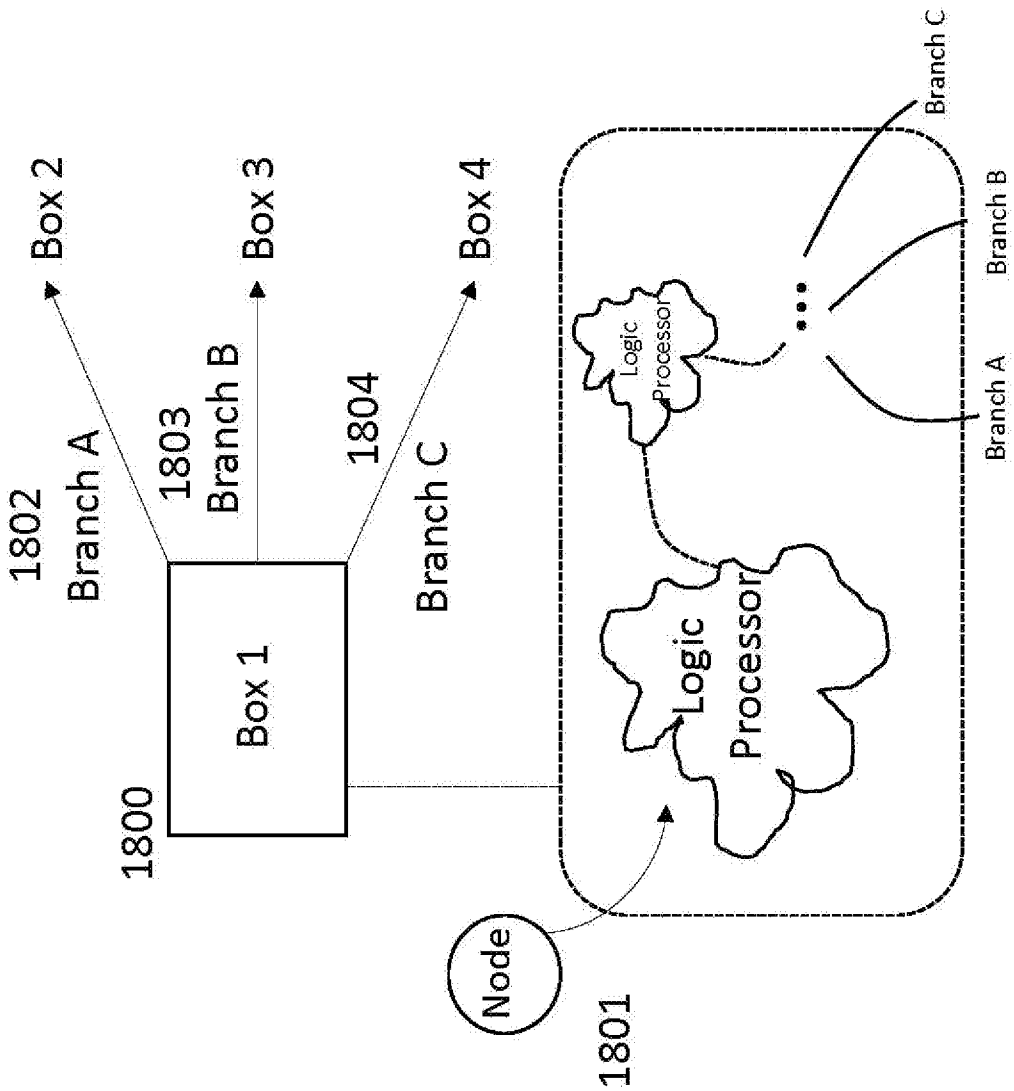

Box 2
Box 3
Box 4
1802
Branch A
1803
Branch B
1804
Branch C
Box 1
1800
Node
1801
Logic Processor
Logic Processor
Branch C
Branch B
Branch A

FIG. 18

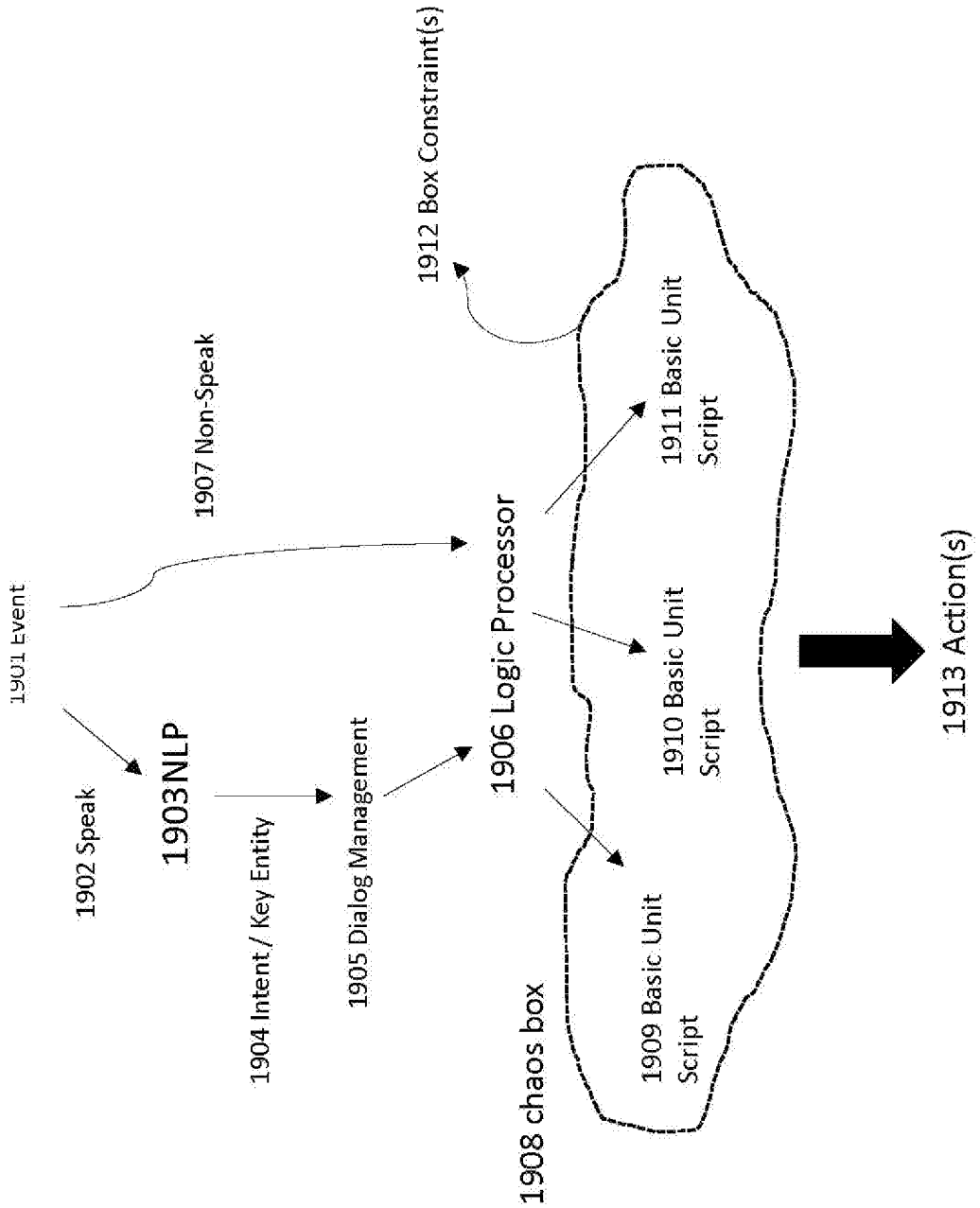

1901 Event
1902 Speak
1903NLP
1904 Intent / Key Entity
1905 Dialog Management
1906 Logic Processor
1907 Non-Speak
1908 chaos box
1909 Basic Unit Script
1910 Basic Unit Script
1911 Basic Unit Script
1912 Box Constraint(s)
1913 Action(s)

FIG. 19

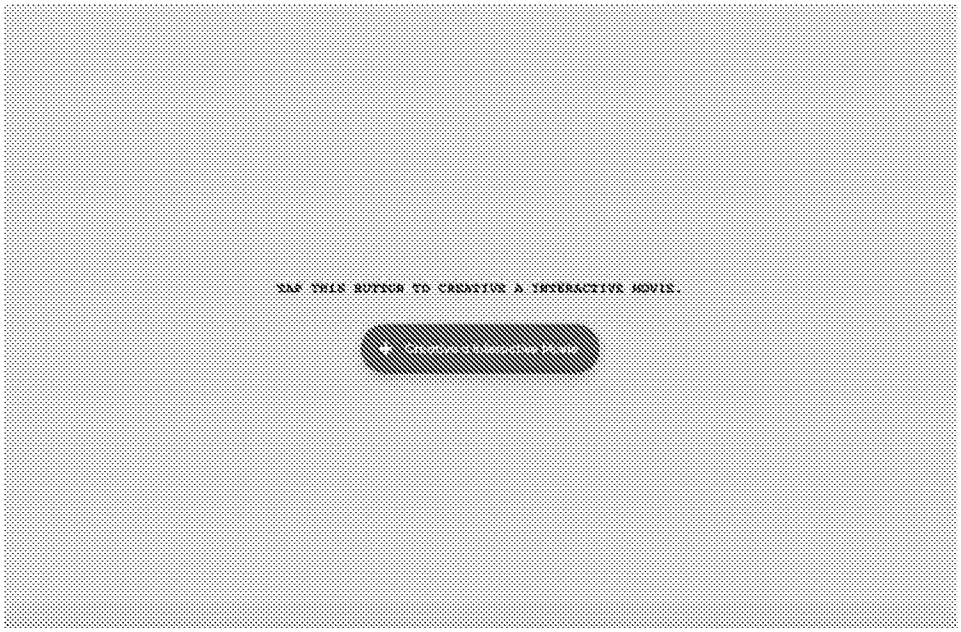


FIG. 20A

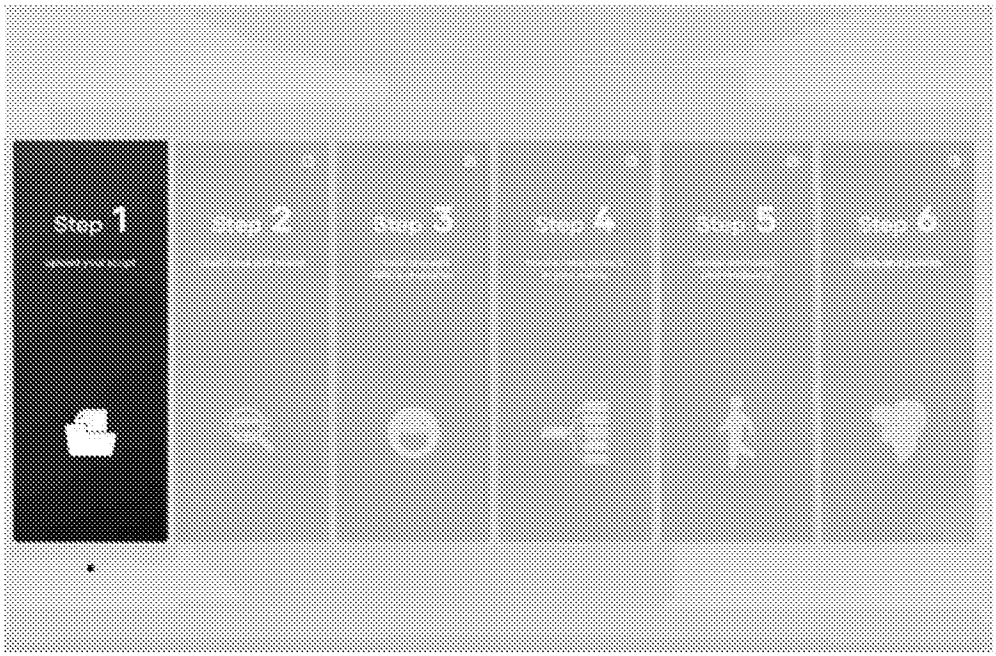


FIG. 20B

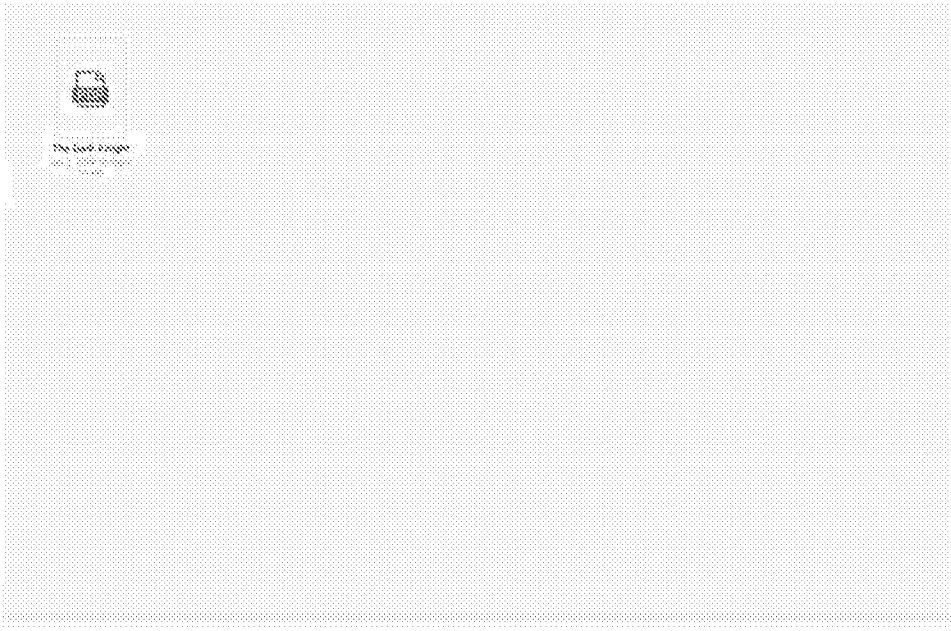


FIG. 20C

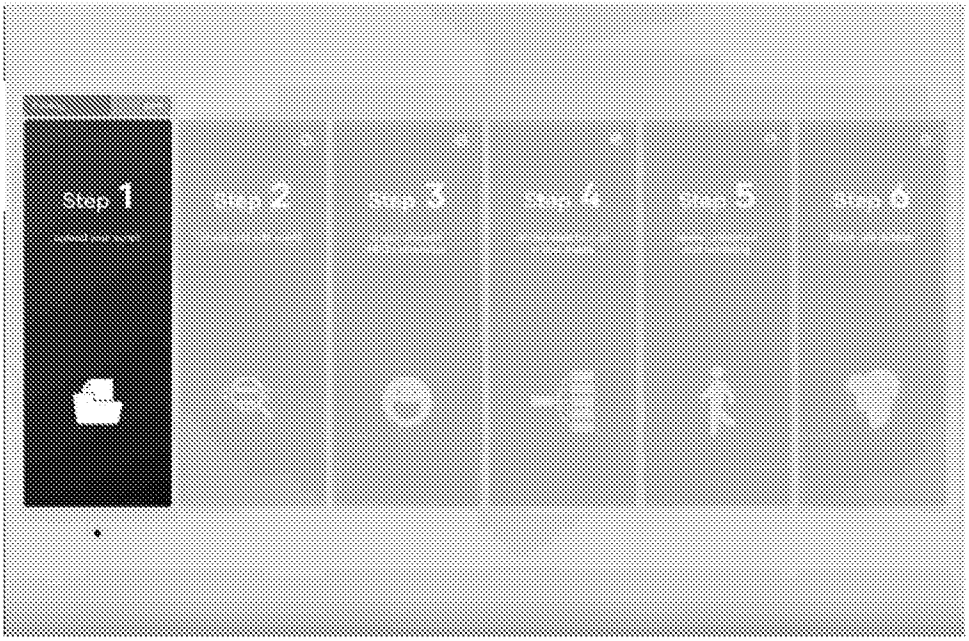


FIG. 20D

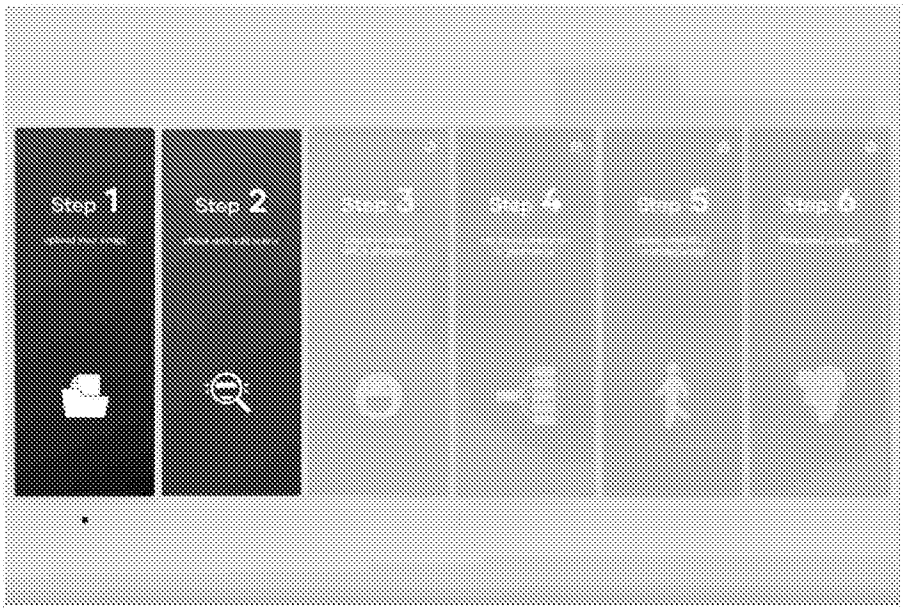


FIG. 20E

FIG. 20F

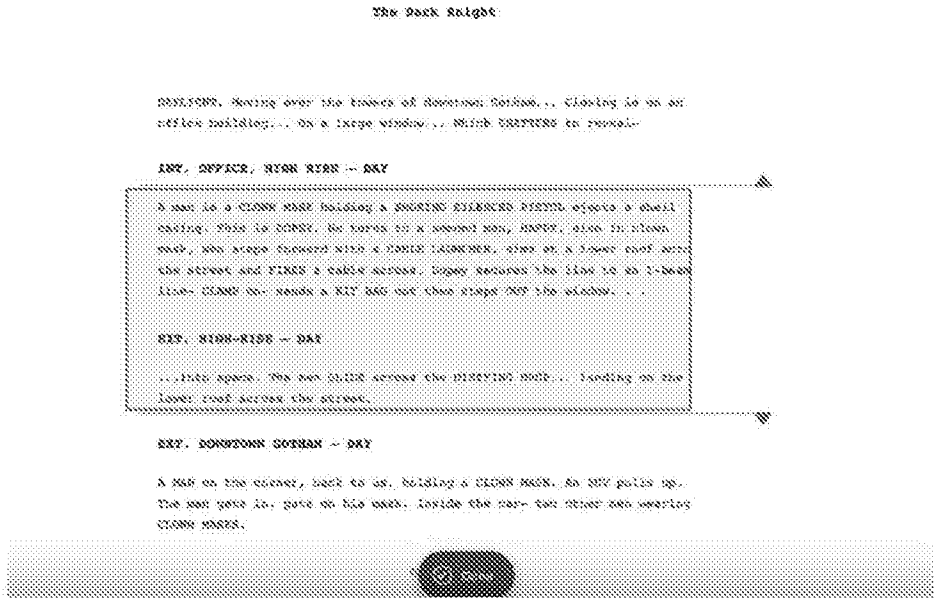


FIG. 20G

FIG. 20H

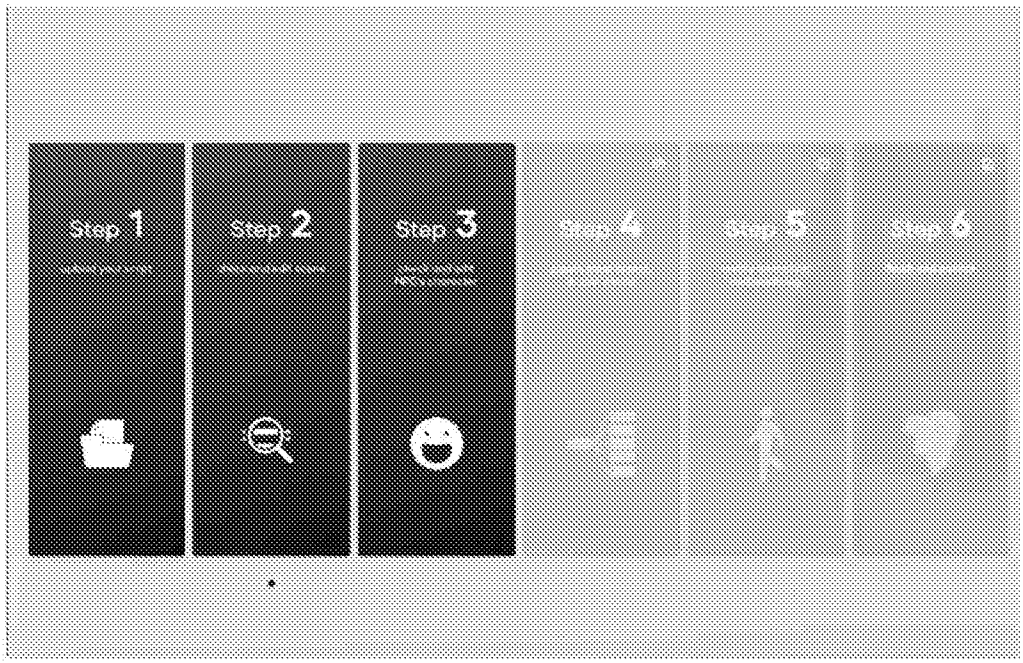


FIG. 20I

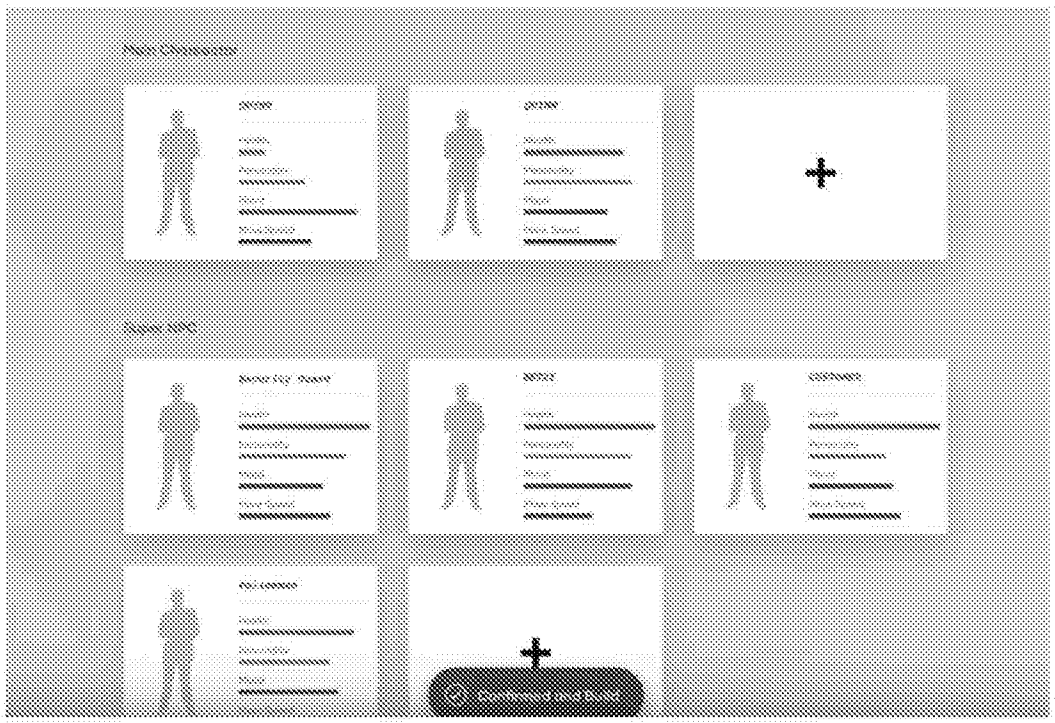


FIG. 20J

FIG. 20K

FIG. 20L

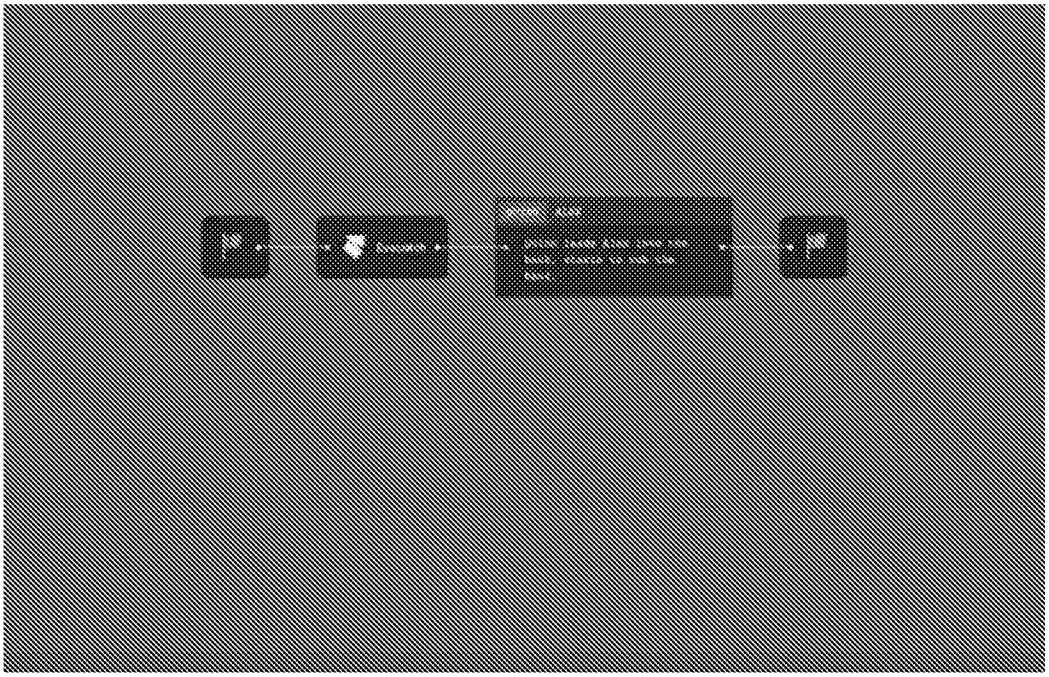


FIG. 20M

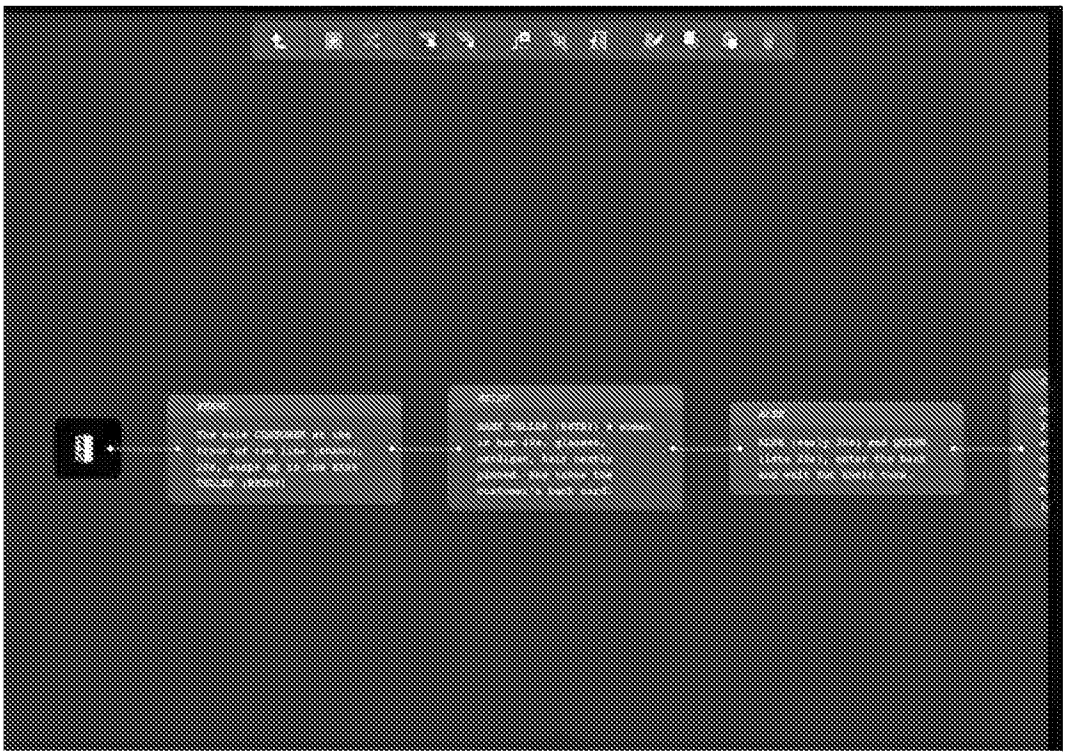


FIG. 20N

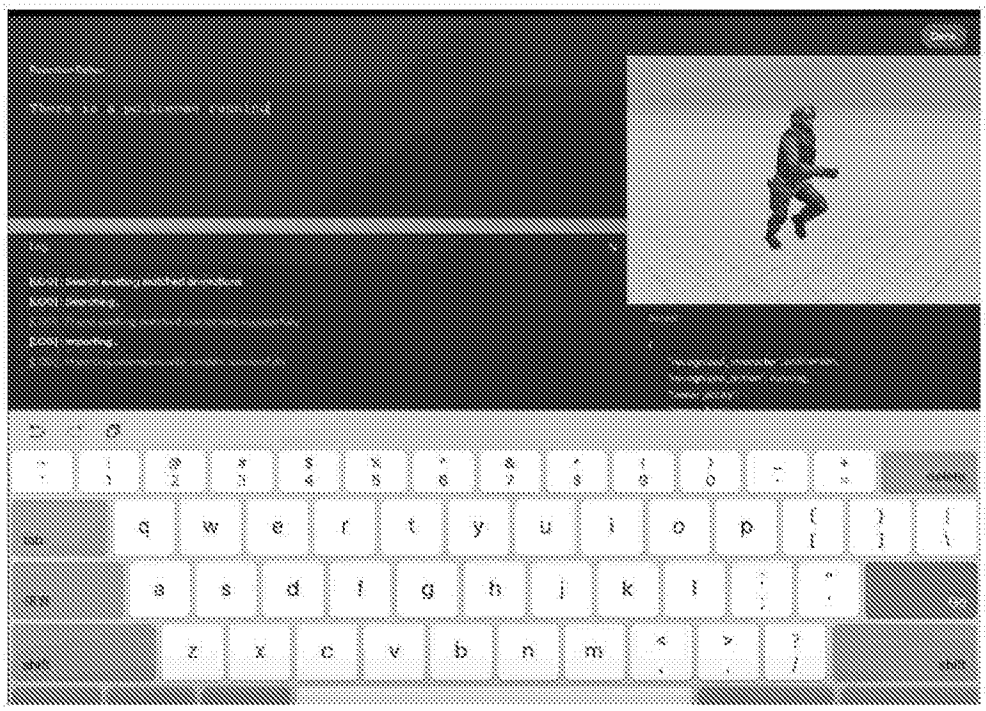


FIG. 20O

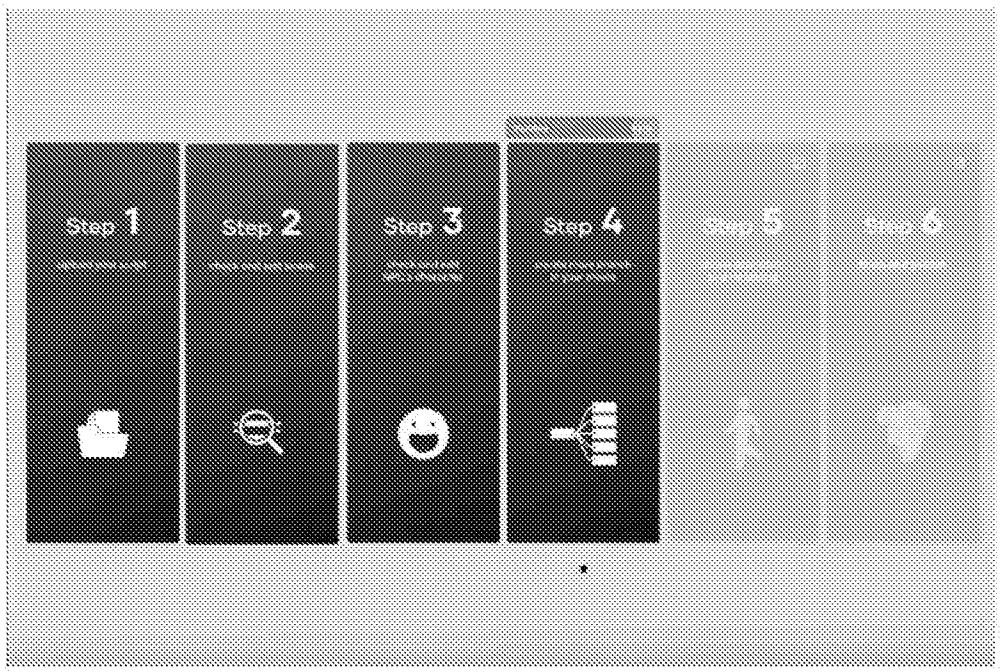


FIG. 20P

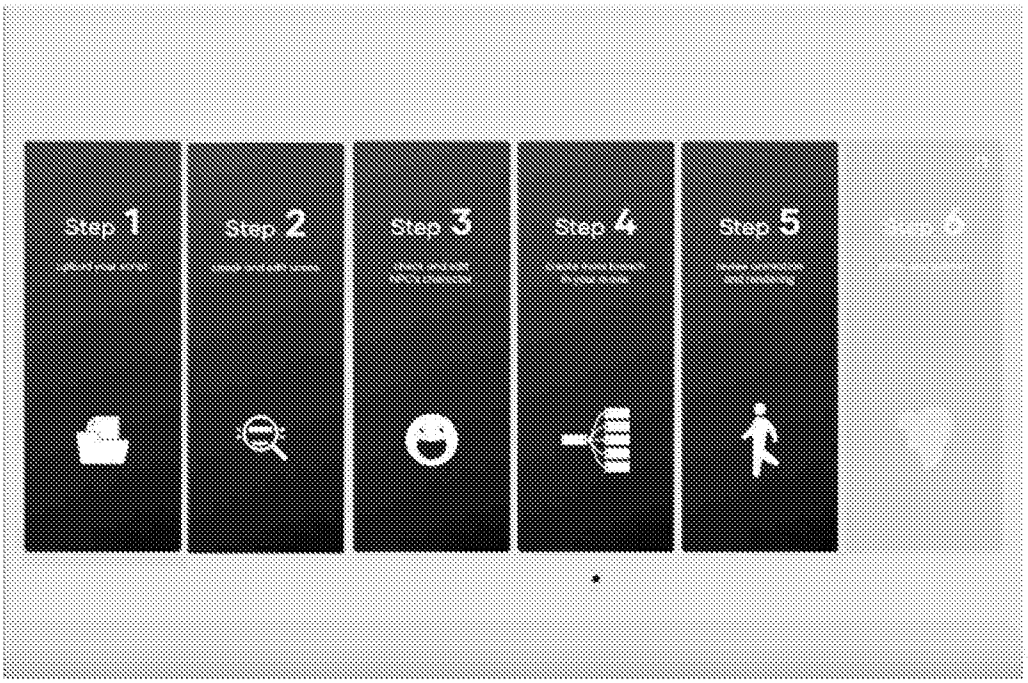


FIG. 20Q

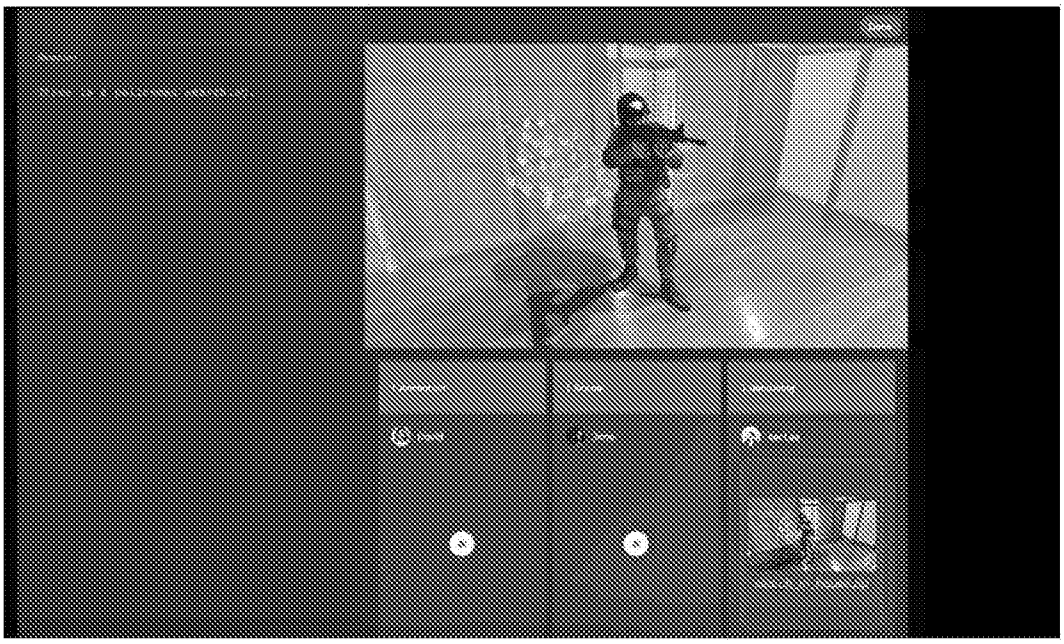


FIG. 20U

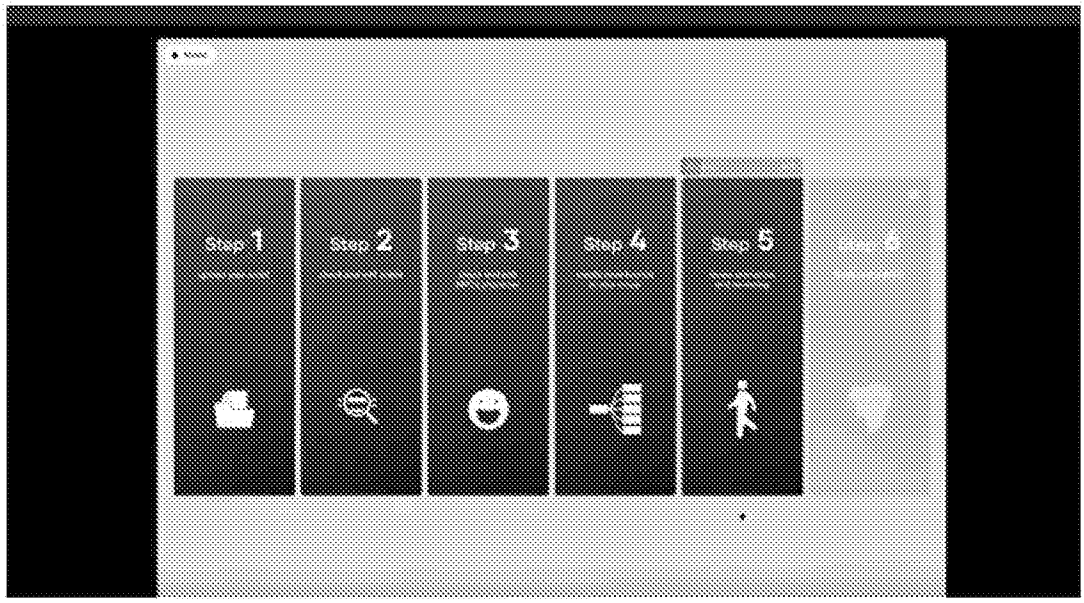


FIG. 20V

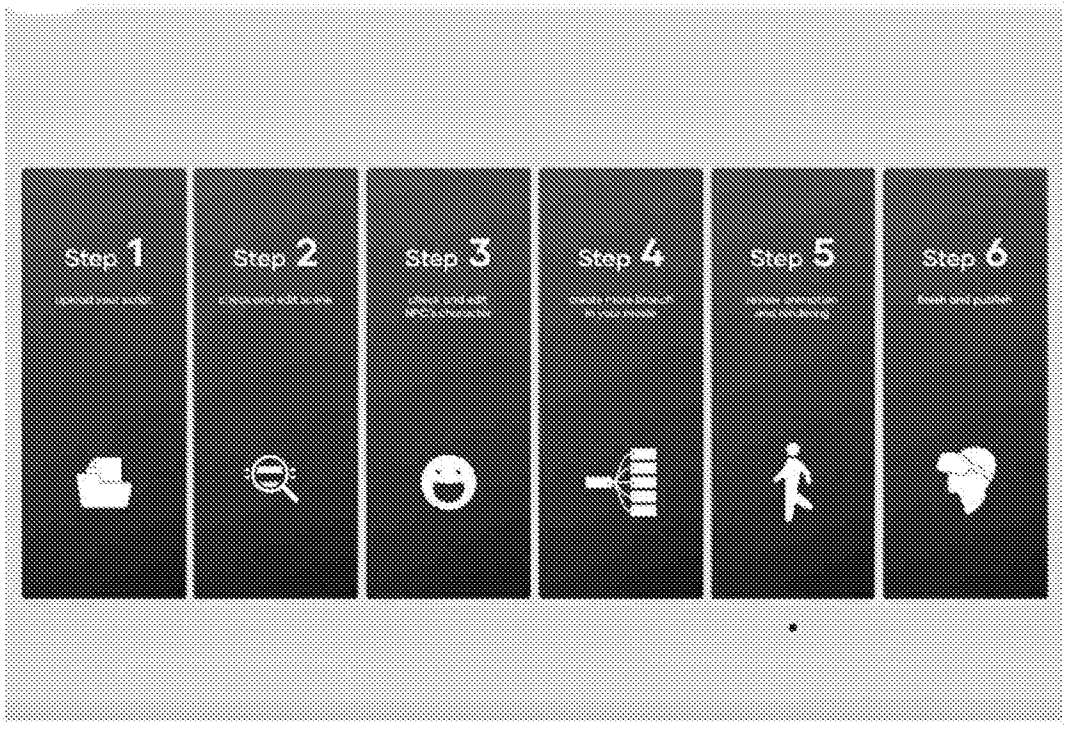


FIG. 20W

FIG. 20X

FIG. 20Y

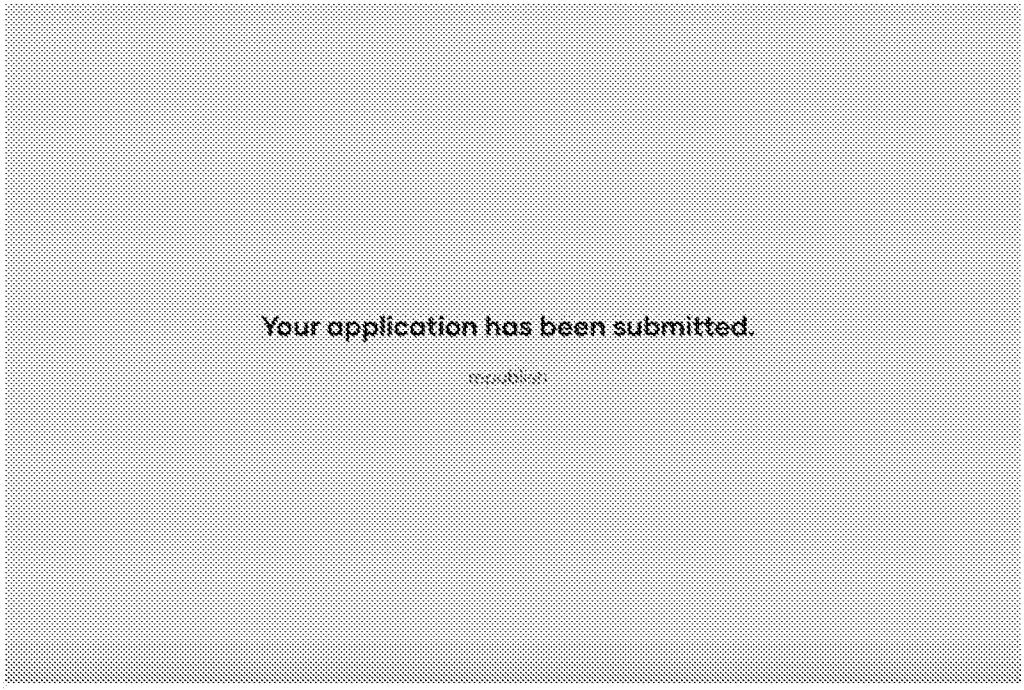
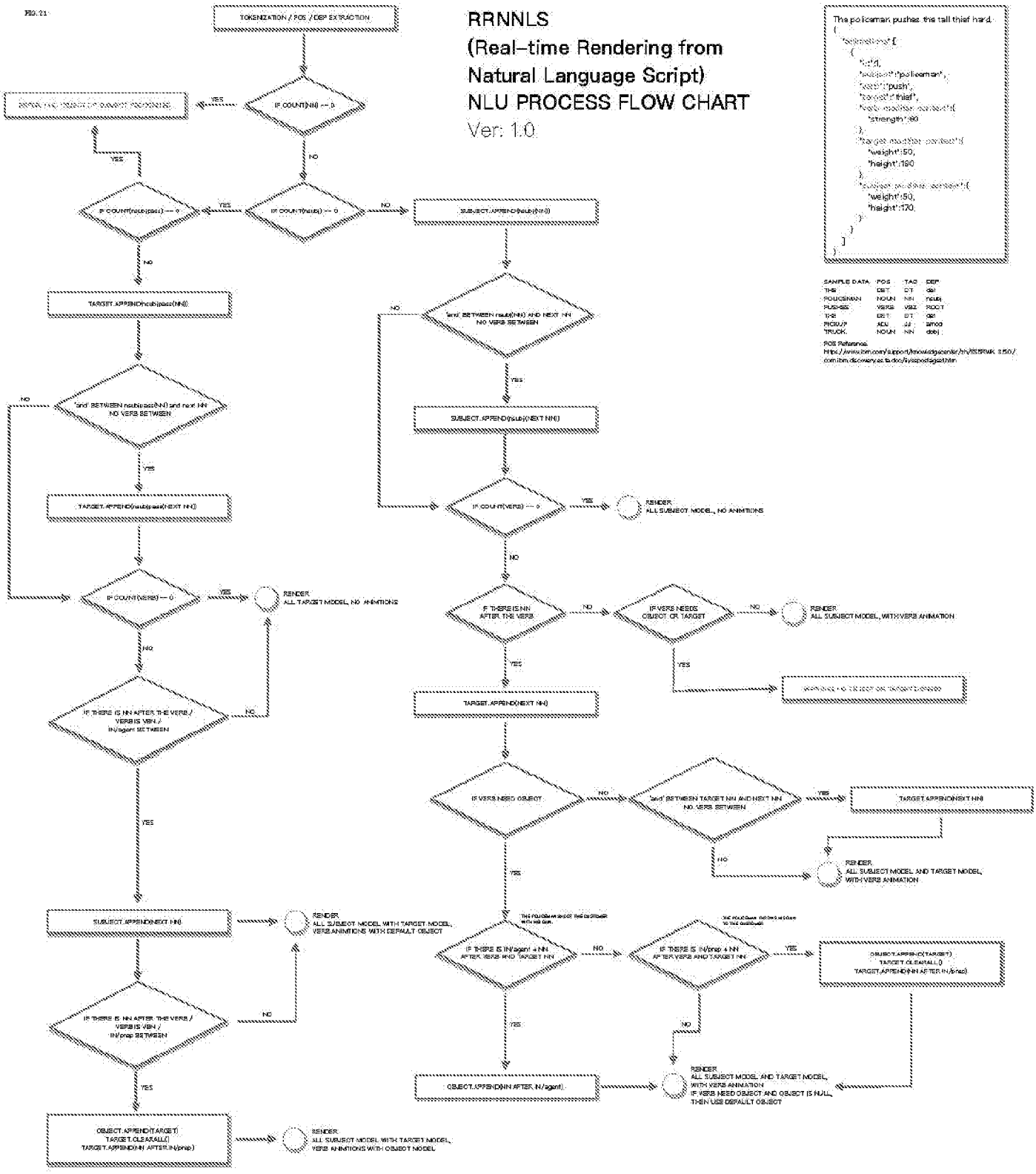


FIG. 20Z


RRNNLS
(Real-time Rendering from Natural Language Script)
NLU PROCESS FLOW CHART
Ver: 1.0
TOKENIZATION / POS / DEP EXTRACTION
IF COUNT(NN) == 0
YES
NO
IF COUNT(nsubjpass) == 0
IF COUNT(nsubj) == 0
SUBJECT.APPEND(nsubj(NN))
TARGET.APPEND(nsubjpass(NN))
'and' BETWEEN nsubjpass(NN) and next NN NO VERB BETWEEN
'and' BETWEEN nsubj(NN) AND NEXT NN NO VERB BETWEEN
TARGET.APPEND(nsubjpass(NEXT NN))
SUBJECT.APPEND(nsubj(NEXT NN))
IF COUNT(VERB) == 0
RENDER ALL TARGET MODEL, NO ANIMTIONS
RENDER ALL SUBJECT MODEL, NO ANIMTIONS
IF THERE IS NN AFTER THE VERB / VERB IS VBN / IN/agent BETWEEN
IF THERE IS NN AFTER THE VERB
IF VERB NEEDS OBJECT OR TARGET
RENDER ALL SUBJECT MODEL, WITH VERB ANIMATION
TARGET.APPEND(NEXT NN)
IF VERB NEED OBJECT
'and' BETWEEN TARGET NN AND NEXT NN NO VERB BETWEEN
TARGET.APPEND(NEXT NN)
RENDER ALL SUBJECT MODEL AND TARGET MODEL, WITH VERB ANIMATION
SUBJECT.APPEND(NEXT NN)
RENDER ALL SUBJECT MODEL WITH TARGET MODEL, VERB ANIMITIONS WITH DEFAULT OBJECT
IF THERE IS NN AFTER THE VERB / VERB IS VBN / IN/prep BETWEEN
IF THERE IS IN/agent + NN AFTER VERB AND TARGET NN
IF THERE IS IN/prop + NN AFTER VERB AND TARGET NN
OBJECT.APPEND(TARGET) TARGET.CLEARALL() TARGET.APPEND(NN AFTER IN/prop)
OBJECT.APPEND(TARGET) TARGET.CLEARALL() TARGET.APPEND(NN AFTER.IN/prep)
RENDER ALL SUBJECT MODEL WITH TARGET MODEL, VERB ANIMITIONS WITH OBJECT MODEL
OBJECT.APPEND(NN AFTER, IN/agent)
RENDER ALL SUBJECT MODEL AND TARGET MODEL, WITH VERB ANIMATION IF VERB NEED OBJECT AND OBJECT IS NULL, THEN USE DEFAULT OBJECT
The policeman pushes the tall thief hard.
SAMPLE DATA POS TAG DEP
THE DET DT det
POLICEMAN NOUN NN nsubj
PUSHES VERB VBZ ROOT
THE DET DT det
POLICE ADJ JJ amod
TRUCK. NOUN NN dobj
POS Reference
https://www.ibm.com/support/knowledgecenter/en/SSERMK_8.5.0/com.ibm.discovery.es.ta.doc/iiysposttagset.htm

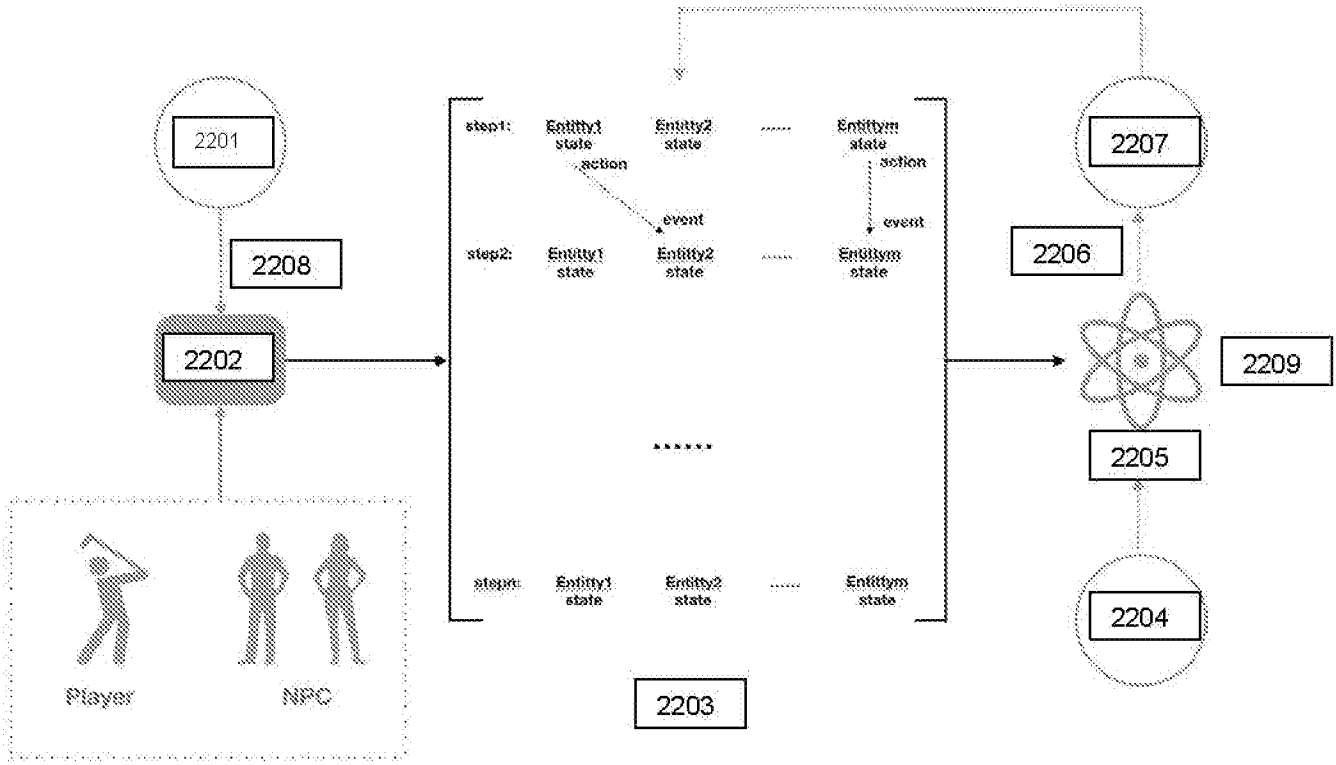

2201
2208
2202
Player
NPC
step1:
Entitty1 state
Entitty2 state
Entittym state
action
event
step2:
Entitty1 state
Entitty2 state
Entittym state
stepn:
Entitty1 state
Entitty2 state
Entittym state
2203
2207
2206
2209
2205
2204

FIG. 22

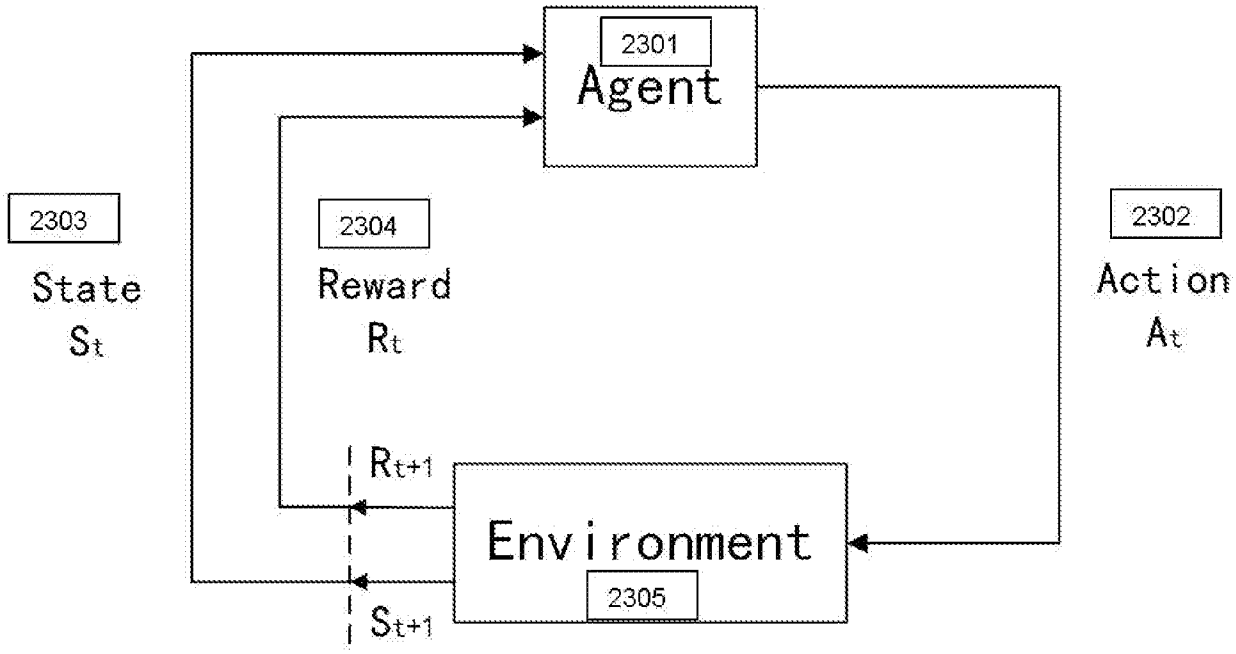

2301
Agent
2303
State
S_t
2304
Reward
R_t
2302
Action
A_t
R_{t+1}
Environment
2305
S_{t+1}

FIG. 23

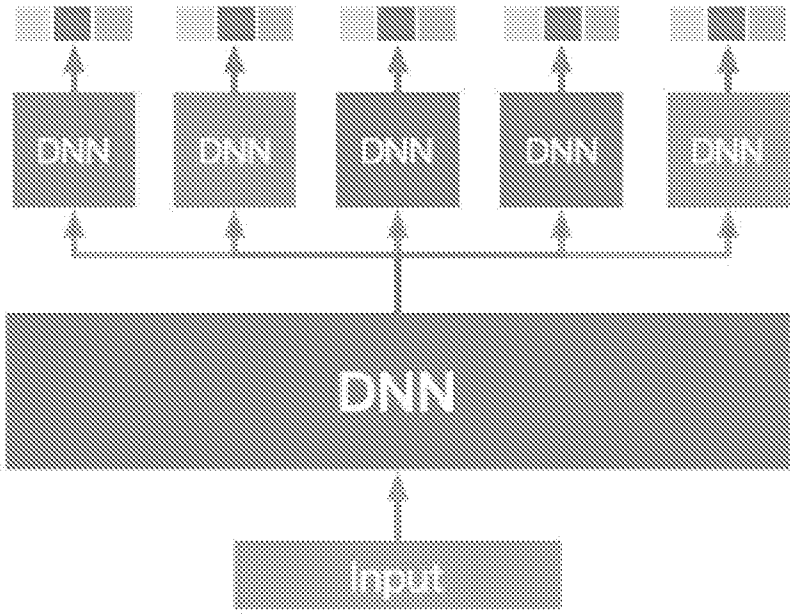

DNN
DNN
DNN
DNN
DNN
DNN
Input

FIG. 24

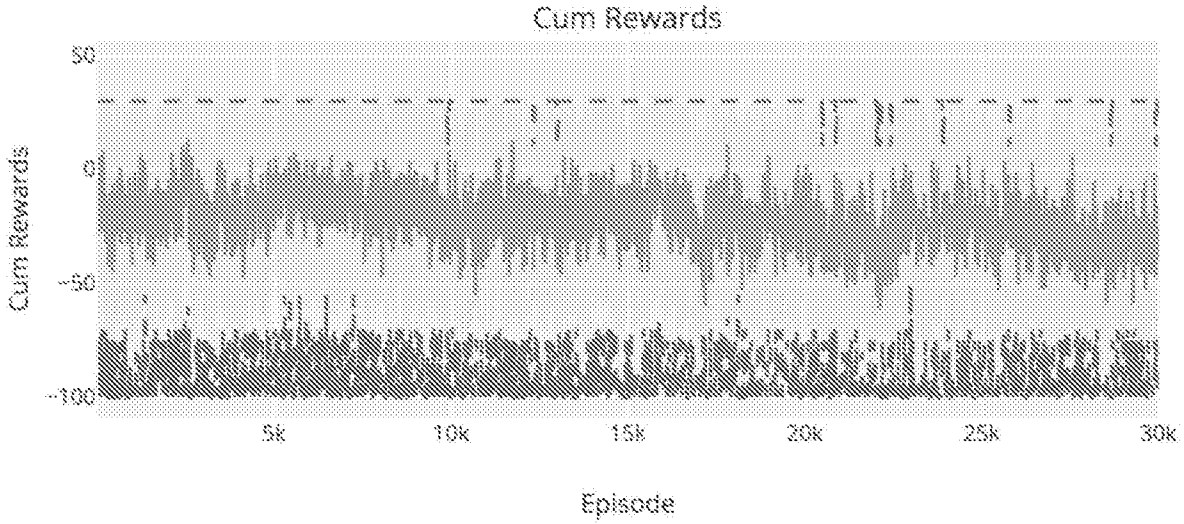


FIG. 25A

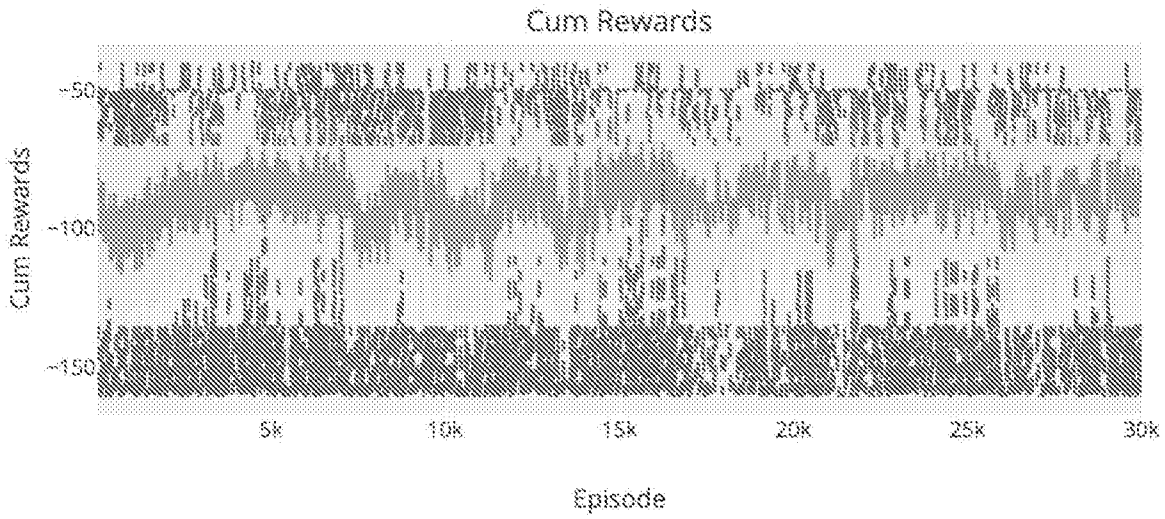


FIG. 25B

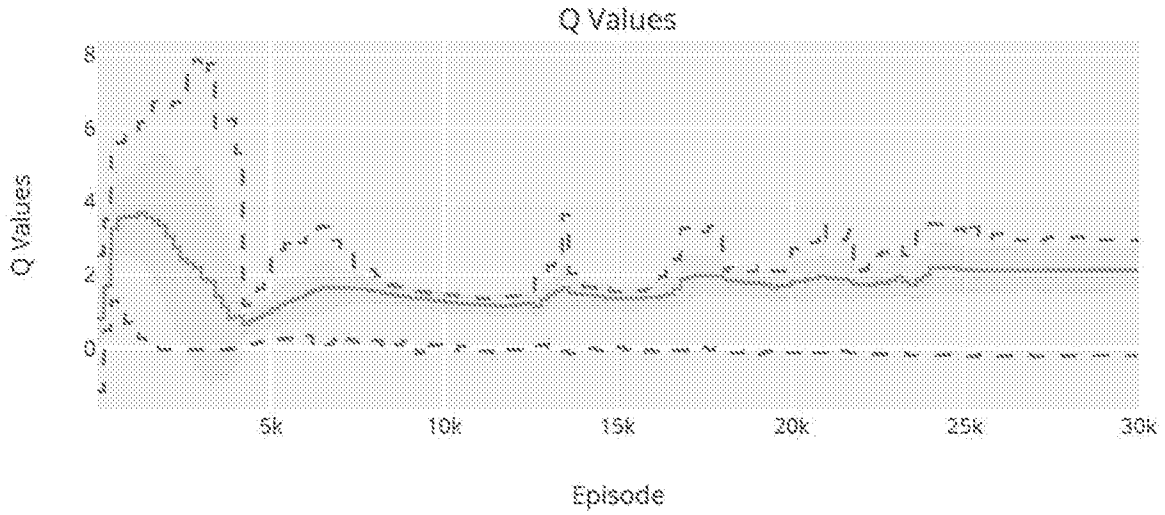


FIG. 26A

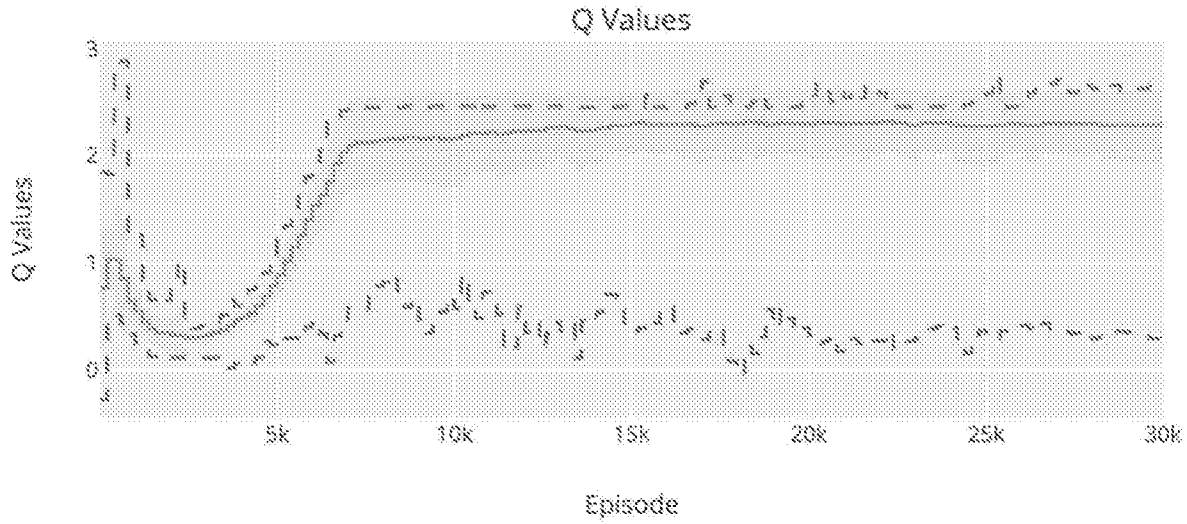


FIG. 26B

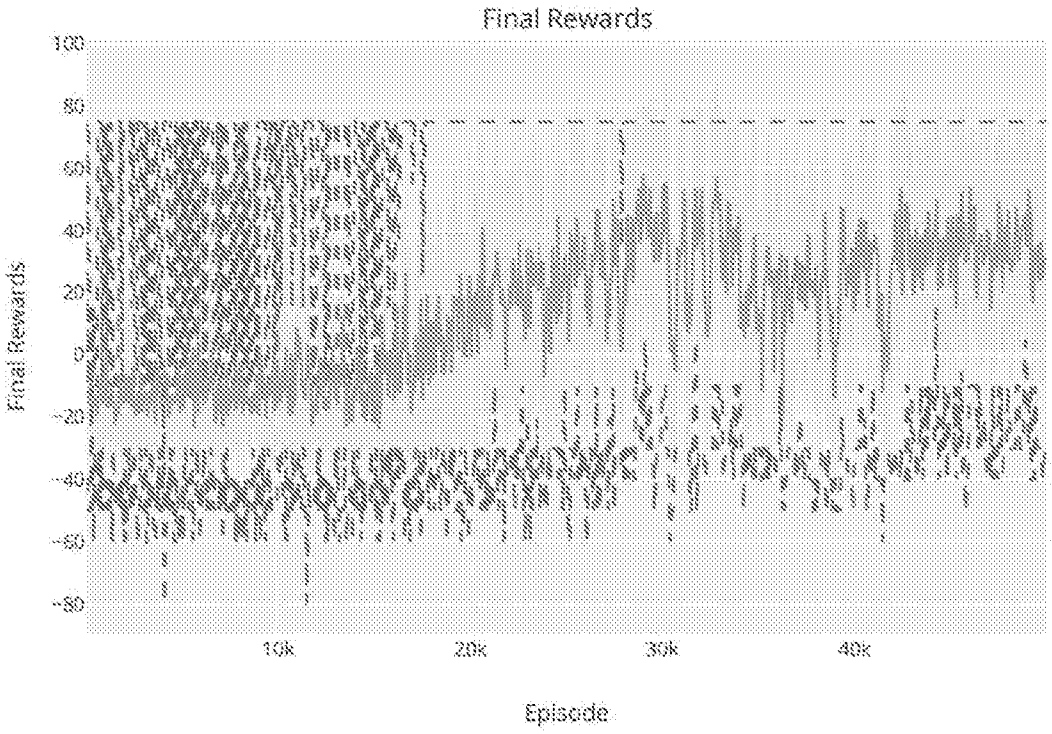

Final Rewards
100
80
60
40
20
0
-20
-40
-60
-80
Final Rewards
10k
20k
30k
40k
Episode

FIG. 27


Current Sate 1
Current Sate 2
Current Sate 3
Agent 1
Agent 2
Agent 3
Agent 4
Agent 5

FIG. 28

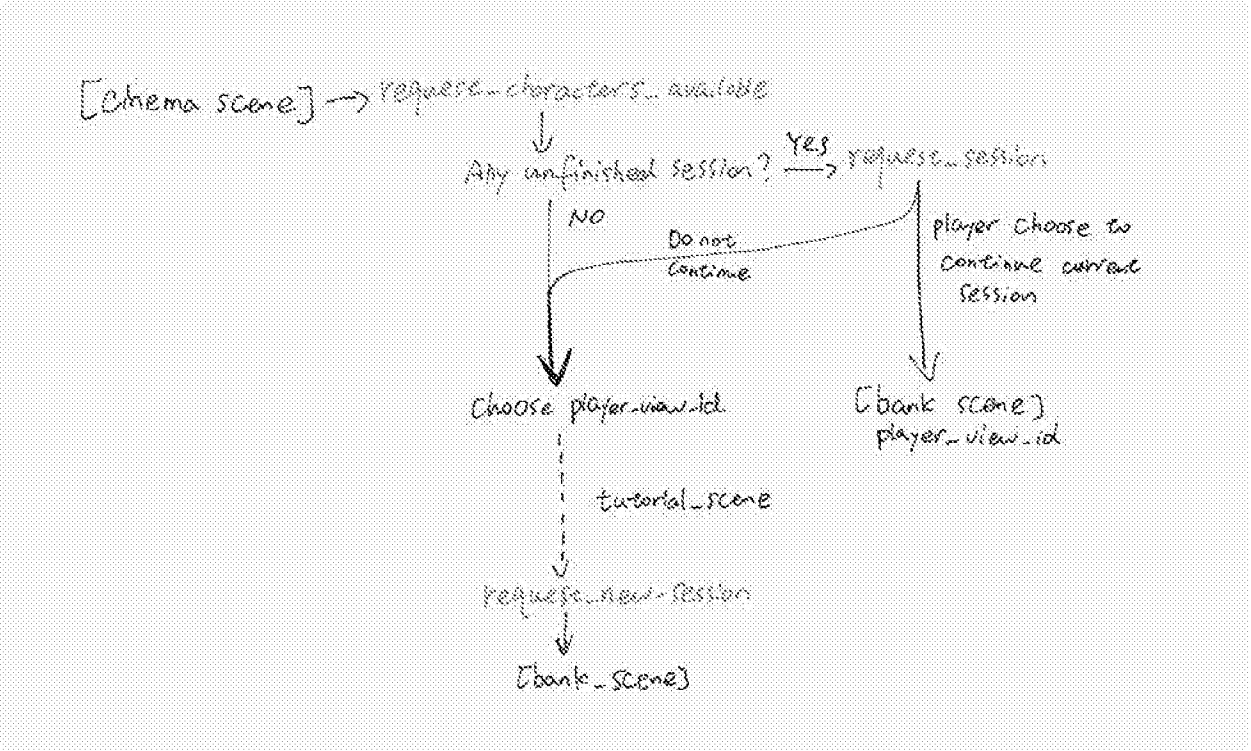

[Schema scene] → request_characters_available
Any unfinished session?
Yes
request_session
NO
Do not continue
player choose to continue current session
Choose player_view_id
[bank scene]
player_view_id
tutorial_scene
request_new_session
[bank_scene]

FIG. 29

INTERNATIONAL SEARCH REPORT

International application No.
PCT/US2020/022670

A. CLASSIFICATION OF SUBJECT MATTER

IPC(8) - A63F 13/56; G06F 3/01; G06N 20/00; G06T 19/20 (2020.01)
CPC - A63F 13/56; G06F 3/011; G06K 9/00671; G06N 20/00; G06T 19/20 (2020.05)

According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)
See Search History document

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched
USPC - 345/8; 345/419; 345/473; 345/633; 706/46 (keyword delimited)

Electronic data base consulted during the international search (name of data base and, where practicable, search terms used)
See Search History document

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X --- Y	US 2013/0229433 A1 (REINCLOUD CORPORATION) 05 September 2013 (05.09.2013) entire document	1, 3, 4, 6, 9, 11, 12, 14, 17 --- 2, 5, 7, 8, 10, 13, 15, 16, 19
X --- Y	US 8,228,335 B1 (JACOB et al) 24 July 2012 (24.07.2012) entire document	18 --- 7, 8, 15, 16
Y	US 2002/0168621 A1 (COOK et al) 14 November 2002 (14.11.2002) entire document	2, 10
Y	US 8,326,626 B1 (PETTAY et al) 04 December 2012 (04.12.2012) entire document	5, 13
Y	WO 2009/055929 A1 (XTRANORMAL TECHNOLOGIE INC.) 07 May 2009 (07.05.2009) entire document	8, 16, 19

☐ Further documents are listed in the continuation of Box C. ☐ See patent family annex.

* Special categories of cited documents:
"A" document defining the general state of the art which is not considered to be of particular relevance
"E" earlier application or patent but published on or after the international filing date
"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)
"O" document referring to an oral disclosure, use, exhibition or other means
"P" document published prior to the international filing date but later than the priority date claimed
"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention
"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone
"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art
"&" document member of the same patent family

Date of the actual completion of the international search	Date of mailing of the international search report
21 May 2020	11 JUN 2020
Name and mailing address of the ISA/US Mail Stop PCT, Attn: ISA/US, Commissioner for Patents P.O. Box 1450, Alexandria, VA 22313-1450 Facsimile No. 571-273-8300	Authorized officer Blaine R. Copenheaver PCT Helpdesk: 571-272-4300 PCT OSP: 571-272-7774