Analyzing Computer Game Strategies through Visual Techniques

Mircea Catalin Catana, Dorian Gorgan
Computer Science Department, Technical University of Cluj-Napoca
Str. G. Baritiu 28, 400027, Cluj-Napoca, Romania
mircea.catalin.catana@gmail.com, dorian.gorgan@cs.utcluj.ro

ABSTRACT
The paper presents and analyzes an application that can emulate different types of strategy games and provide significant specifications as to how the strategy impacts the outcome of the player experience. This application is meant to be used as a tool both by game developers, which can use it to calibrate their products and also by the game players, who will want to use it as a means to improve their strategies. The research explores the concept of strategy in games and tries to define a metric for evaluating this concept. Due to the complexity of videogames, the computation of every outcome for every possible decision a player can make is impossible, so the research proposes a method of combining the human familiarity with the game concept and the processing power of the computer to search the wide solution space. The paper analyses a concrete example by building a test bed for the tower defense type games.

Author Keywords
Game Strategy; Visual Analytics; Visual Techniques; Tower Defense Games.

ACM Classification Keywords
H.5.2 User Interfaces, I.3.7 Three-Dimensional Graphics and Realism, I.6.8 Types of Simulation.

INTRODUCTION

Video Games have been around for a long time, dating to the early 50's as part of computer science research, however it is not until the 90's when computers began to spread and be used by the general public that games have started to flourish. Where in the early days of game development a person could do most of the work by himself, highly specialized teams are now working on specific aspects of the game like: gameplay, user interface (UI), game design, etc.

Along the road games started to be categorized by their creators and players in types related to their content, play style and view of their simulated environment. Some of these categories include: 2D and 3D worlds, platformers, puzzles, role playing games (RPGs), etc. Out of all these, this paper focuses on games in which the player must devise a strategy as the core part of the gameplay in order to win the game. These are usually called strategy games and can be either turn based or real time.

In turn based games, each player takes turns in a predefined order to perform actions which have impact in the game world. A classic example of a turn based game is chess in which the white and black players move a piece per turn until an end game situation is achieved.

In real time strategy (RTS) games [9] the players interact with the world at the same time and practically "race" each other to the end of the game. These types of games are more challenging as players are not only forced to play a better strategic game than their opponent but they also have to make decisions fast in order to be quicker than the other players.

Our research mainly deals with tower defense games as a demonstration of the applicability of the visual analysis techniques used, but future releases could cover a wider array of game types. Tower defense games are a particular type of strategy game in which the user plays against a computer in single player mode. The player has to defend a safe point by building different types of turrets that shoot at the computer units. The computer's objective is to reach the player's safe point with its units. The units follow a predefined path from a spawn point to the player safe point and are generally generated in waves of increasing difficulty. The challenge and strategy lies in how the human player decides to spend his resources in order to build turrets which will defend his safe point. After each computer unit kill, the human player is awarded a small amount of resources which he will later be able to spend on building more powerful turrets. The game allows for a minimum number of computer units to reach the safe point before the player loses the game. If however the player manages to defeat all the enemy waves before that threshold is reached he wins.

In creating games, one frequent problem is that developers have to take great care and spend a lot of time balancing the properties of their software so that the experience of the player is enjoyable. This means tweaking all the variables in a way such that the game is not extremely easy to play, but also not very difficult or impossible. This process can be a daunting task, but an absolutely necessary one if the game is to be successful. Currently, there is no general purpose software which can aid in this process. Some gaming companies develop a small array of in-house tools that help game designers in this regard, but these are small

tools that are usually custom built for each game individually.

This paper proposes a method of addressing the previously mentioned issues by building a software tool which is able to simulate the conditions of a computer game, analyze the way a strategy is carried out and provide insight into ways of improving gameplay. This can be beneficial to both developers by providing them with a tool that can aid the process of calibrating the game difficulty levels and professional gamers which can use the software to improve the strategies they use in competitive games as well.

Videogames are highly complex systems with a lot of variables which means that we cannot simply compute every possible outcome for all the sets of inputs and just choose the best case. The method described in this paper proposes to combine the computation power of a computer with the analytical thinking and decision making capabilities of a human being. The main goal of this research is to develop a software application that can analyze strategies used in tower defense games and provide the user with visual guidance in developing a better strategy for a specific game input.

If we are to analyze all the possible combinations a user can choose to play during a game we have to think of the variables of the game as an n-dimensional solution space. Such a space where the axes have values like turret damage and position on the grid is very hard to be interpreted and visualized by a human being. A difficult task was to come up with a visual representation that would provide a view of this system from a perspective in which the user could easily understand the parameters and the optimal values for them. The paper proposes a visualization technique that makes use of the projections on the axes of the n-dimensional space.

The paper is structured as follows: Introduction section provides an overview of computer games and introduces the reader to the concept of strategy in games. This section also provides the motivation behind building this software product. Section 2 is a study of the work done by others which provides a base for this research. It treats strategy in video games and visualization techniques independently. Next section covers the theoretical analysis of the solutions, providing insight into the design of the application and the technologies used during development. The solutions are evaluated and validated by practical experiments. Last section draws conclusions and future directions.

RELATED WORKS

Game engines are highly complex software systems that are designed for the development of videogames. The concept is relatively new as games before the mid 90's were built from ground up every time. Jason Gregory marks in his "Game Engine Architecture" book [1] that the concept of game engines started to form around games like *id Software*'s Doom. The game provided a clear separation between core functionality and specific game aspects like custom assets or gameplay rules. By the late 90's game developers identified certain needs any game would have regardless of its contents and started building software that would manage these needs but that could also be reusable in other projects. Some of these components include a rendering engine, user input handling, a sound framework, and so on. A separation between the layers handling game logic and the lower layers handling the hardware and general purpose systems started forming in all the games since then. This eventually became known as the game engine. It is worth noting though, that not even today's modern game engines can provide a clear separation between them and the game.

An important point in the history of game engines was introduced by game moding. The concept of a game mod or modification was introduced with the rise of games like *id Software*'s Quake Arena and *Epic Games*' Unreal Tournament. In these games the engines were made highly customizable via scripting languages like Quake C. These moding capabilities allowed anyone who owned a license to build additional content for the original game or even build a new game. Shortly a large community was built around this concept and this facilitated the separation of the game engines from the actual games.

Nowadays there are a high number of different game engines that are licensed separately from any specific game. These engines have reached a point where anyone can pick them up and build any game they have in mind without worrying about how to implement every system that a videogame would need in order to run. There is enough free software to cover most of the needs of a developer, so the problem becomes more of choosing the right tool rather than not having access to it. An example of such commercially available game engines that are at our disposal now include Epic Games' Unreal Engine 4, which has features that mainly target FPS games and Unity Technologies' Unity3D engine.

Game engines are usually made from a runtime component and several tools that can work along it. The architecture is built in layers where upper layers depend on lower ones but not the other way around so as not to create circular dependencies and to promote low coupling between the systems.

One of the most important factors in whether someone keeps playing a videogame is the challenge level of it. The challenge level is directly linked to the engagement the user experiences while playing. Fraser et al. [2] presents data from an experiment in which they analyze how different game factors affect a player's performance in a game. Their paper directly relates to some aspects of this work, as they also build a test bed for a game with the goal of identifying which game factors affect gameplay. In the same way, the software application described in our paper can be used to examine in more detail these factors and their impact.

Besides tweaking parameters to different levels of difficulty, modern games are starting to use the data they gather during a play session and combine it with artificial intelligence (AI) to make the game difficulty level adapt in time. Every game has a learning curve and even though initially a low difficulty level might be suited to a certain player, he could learn the game and its mechanics rather quickly in which case a more difficult level should be ideal. So, if there is no difficulty adapting mechanism installed, the situation could rapidly converge to the initial problem.

In the early games, this need for increasing difficulty was mainly hard-coded by the developers in the design of the game levels. Each successive level would present a harder puzzle to solve or just more of the same difficulty obstacles to get pass by. This technique is still used today, but no longer enough. The racing game Forza, for example, uses the data gathered during online gameplay sessions to adapt the single player AI driving style. This means that as you climb higher in ranks during online play and get better, the single player AI will also be close to your ranking and driving performance. Moreover, if the player usually plays online versus his friends, the AI style will start to mimic his friend's driving style, thus enabling a user to have a connection with his friends even in offline mode.

Other games take an "in the middle" approach and predefine a set of parameters which will be used as a difficulty step during gameplay. If the player performs well and is starting to overcome the current level with ease, the difficulty level will be dynamically incremented to the next, if the player has trouble overcoming the current setting it will be dimmed down. Most games use this approach as more often than not, the complexity of the game is too great to build a fully adaptive AI system.

When discussing game difficulty and challenge in a strategy game, the paper "Exploring Design Features for Enhancing Players' Challenge in Strategy Games" [3] found that there are two important aspects one must consider: mental workload and physical effort. It is important to look closely and distinguish between these types of factors at the level of the game type and start working on building the game design around them. An action game for instance has great demand on the perpetual-motor skills of the player, so focusing the gameplay and building the game around this rather than on puzzle solving is essential.

In the context of strategy games, mental workload is directly linked with information availability. In short, the more information a player is given about the state of the game and the actions that are happening around him, the less mental work he has to do. The study has found that not only this is true, but also that mental workload tends to decrease in the case there is extremely few information. That is, players who are given too little information about what's going on tend to stop being challenged as they figure they cannot accomplish anything with what they have at their disposal. This is a very important piece of information for game designers as it tells them game balancing is the key factor for keeping players engaged.

Similarly with the case of mental workload, physical effort follows a similar path. In strategy games, the physical effort is measured in terms of the amount of resources a player has available. So again, in order for a game to be challenging, the amount of resources a game makes available to a player must not be too high neither too low.

Based on this study we can infer that in the case of tower defense games there is a strong relationship between the data the player has about the next waves of enemies (mental workload) and the resources available to spend on turrets (physical effort). Thus striking the right balance in a strategy for a tower defense game lies in the calibration of these two factors.

VISUAL BASED GAME STRATEGY SOLUTION

Game Strategy Solution Overview

The research aims to develop and experiment the techniques of Visual Analytics, through an application for analyzing and developing strategies in videogames. The software should be able to model a general type of game and based on the game's rules provide the user with information about how to improve his strategy. The application is meant to be used by players trying to get better at a game and also by game designers that are trying to calibrate the difficulty levels in the game they are developing. The solution treats videogames as an *n*-dimensional system in which each axis represents one of the variables in the game. Each such variable will have a value domain range defined by the game in question.

A set of equations describing the game rules and how these variables affect the strategy in the game will be devised based on a predefined metric. The metric will be built specific to the game and it will consist of a function that will score a set of fixed game parameters. Different parameter values will be given different scores by the metric function depending on how desirable each situation is compared to the other. This scoring will be done for a reduced set of values and the results will be displayed to the user through visual techniques. Based on the user's expertise in the field and the human ability to quickly analyze data, the user will chose a set of parameters from the ones evaluated and the computer will restart the computation for values in the vicinity of the chosen data. This way the process will repeat itself and converge towards an optimal solution. The stopping point of the process is determined by the user which can choose to go further until there is no visible change in the data, thus signaling that the optimum has been reached or he may opt to end the search earlier and have a partial solution available in a few iterations.

For the particular case of Tower Defense games we will analyze the following 3 aspects of implementation: building

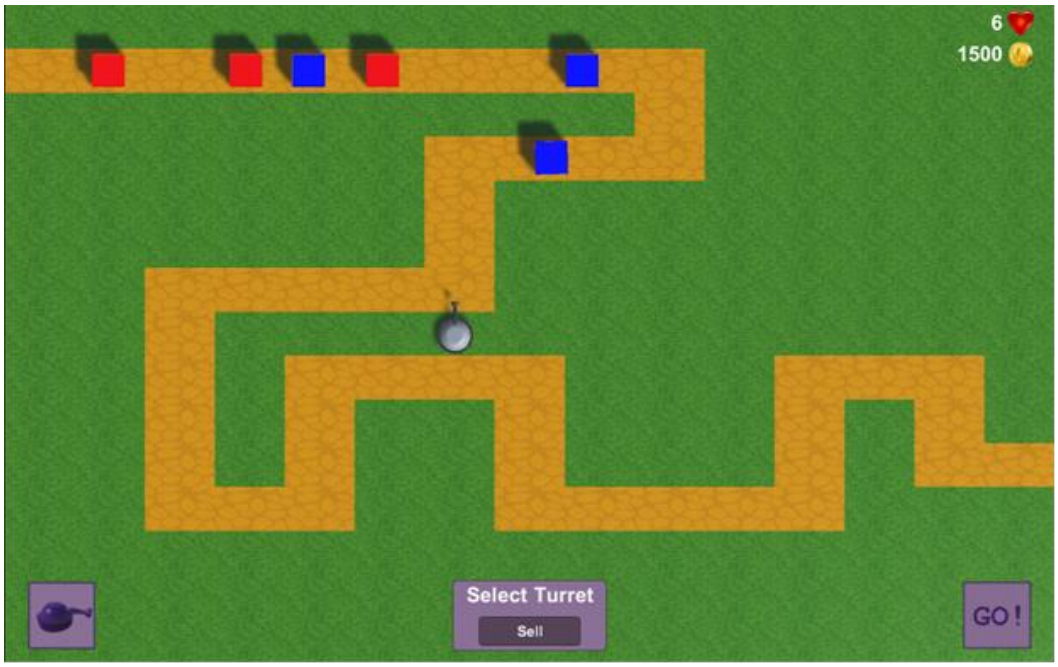

Figure 1. Game graphical user interface.

a simulation of the game, developing the underlying algorithms to analyze the strategy, and building a visualization system that transforms the output of the algorithms into visual data that is easily handled by a human user.

Building the Game

In order to analyze the strategies used, we must first have a working test-bed of the type of game we want to analyze. Since we want to use the application to support as many tower defense games as possible it is better to build a configurable base game ourselves than to try and make a tool which could integrate will all existing software. The implementation of the application is done using the Unity3D engine which allows for easy and quick prototyping of a game (Figure 1). The whole system is stripped to the very basics of tower defense games and is made to be able to model the entities of the game from a configuration file. This ensures that if we have the specification for the game, the system will be able to build a fully functional simulation of it that will come integrated directly with the analysis tools.

Game Evaluation Metrics

Before we start building the algorithms we must first analyze the tower defense specification and determine the metrics for evaluation. The following aspects have been identified as having the most significance in a tower defense strategy:

- Player health at the end of the game
- Resources left unspent
- Difference between total damage dealt and total enemy hit-points
- Total game time

Player health at the end because it directly translates into how many computer enemy units have managed to pass the built towers, thus having more hit-points at the end means a better strategy has been used. Resources left unspent influence the value of the strategy because if you can manage to achieve the same end result spending less resources, it means the strategy used has higher efficiency. The difference in enemy total hit-points and the damage dealt is a subtle way of fine tuning the efficiency of the strategy. Dealing more damage than necessary to the enemies translates into resources that could have been saved on that extra damage.

Finally, total game time is a straightforward unit of efficiency measurement, with the strategy that finishes earlier having the greater score. It is to be noted though, that total game time should be computed only by the sum of the time in which waves of enemies are active and not add the time in which the user is thinking about the strategy as this not affects its value. Thus the final efficiency of the strategy may be calculated as:

$$Efficiency = a*PlayerHealth + b*ResourcesUnspent + c*(TotalDamage - TotalEnemyHealth) + d*TotalTime$$

In order to be able to compute the efficiency then we must know all the above information, but in tower defense games enemies come in waves and you only have information about the waves you have already passed and the current

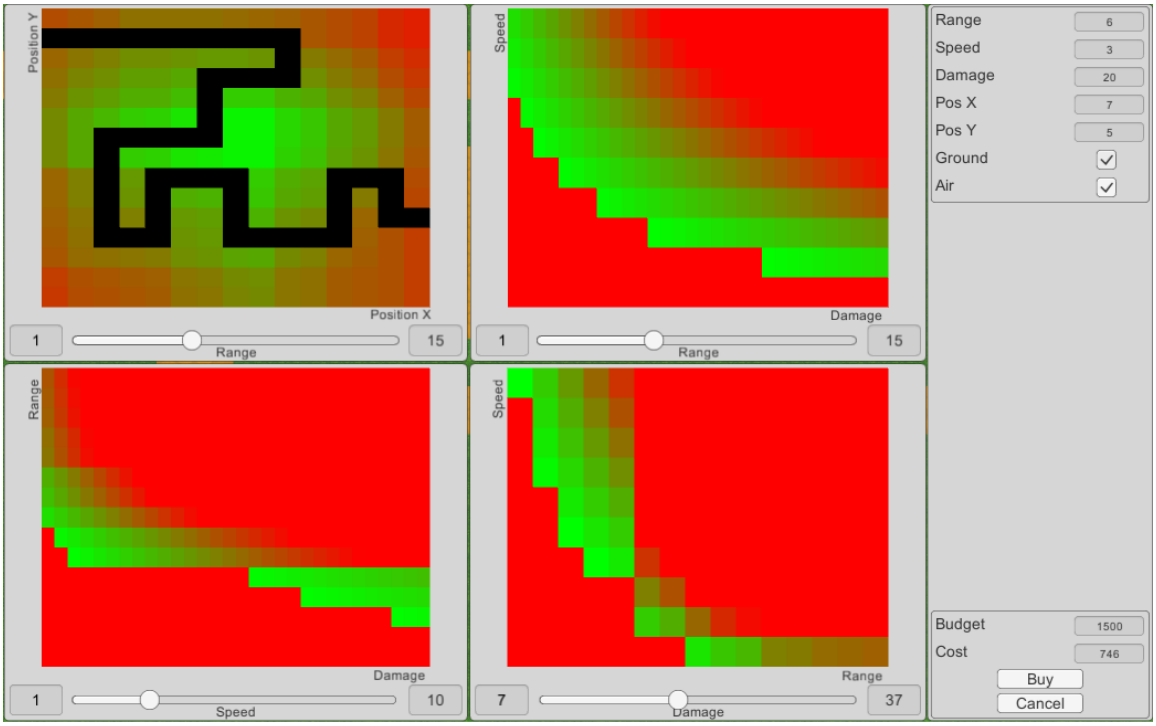

Figure 2. Turret Parameter Graphs.

one. This means that developing a strategy for the game is not possible from the beginning and that at least a play-through must be completed before we can do so.

Developing the Algorithms

The application then provides a method that will not only help the player get to the end of the game and learn the information about the waves as quickly as possible but in the process also learn enough information so that in the end a well-rounded strategy for the whole game can be computed. Thus the paper proposes 2 algorithms:

- Local Optimum – This algorithm will treat only the current wave of enemies with the resources at the player's disposal at that time. It will thus create a turret that has the optimal configuration in order to pass the current wave of enemies.
- Global Strategy – This algorithm will be run at the end of the game, once the information about all the waves is known. It will combine the local optimums developed along the way and generate a strategy considering the whole picture of the game.

To develop the equations for the algorithms we firstly need to know what is the n-dimensional problem space in the tower defense type of game. The following entities are what we must consider, they are split into fixed and variable to indicate that they are given by the specification of the game (fixed), or that they can be influenced by the player's decisions (variable):

Fixed:

Game_World {
    Map,
    Path
}
Enemy {
    Health,
    Speed,
    Damage
}

Variable:

Turret {
    Position,
    Range,
    Damage,
    Firing_Speed
}

The Local Optimum algorithm will try to balance the turret parameters in such a way that the turret cost is minimum, but the turret can also eliminate all the enemies in the current wave. The Range and Position parameters are strongly tied to the Game_World, and the perfect combination of these parameters ensures that the turret can fire for the longest amount of time possible on the enemies. The Range, Damage and Firing_Speed are tied to the Enemy component as they determine the total output damage the turret can achieve in the time the enemies are in the turret's range. The system probes a restrained domain of values for all of the parameters and displays the results to the user using the visual techniques described below.

Building the visualization tool

Up until now we established the metrics and a series of algorithms that can score a combination of the game parameters for a local case. The problem now is to build a

visualization tool that the human user can interact with, and from which he can guide the further search iterations of the algorithm.

It is very hard for people to reason about *n*-dimensional problem spaces especially when $n > 3$ and when the axes do not necessarily represent position. In order to make the user's experience easier we draw inspiration from 3D modeling software where you can see the 3-dimensional space not only from the perspective view but you can also chose to view from 2D viewports that display only 2 of the 3 dimensions at once.

In a similar fashion we choose to represent our problem space through a series of 2D projections where each axis represents the values for one of the turret's parameters (Figure 2). In addition to this representation each of our 2D viewports comes with a slider which is assigned to the domain of a 3rd parameter. By using this slider, the user can see how changes in the slider's parameter values affect the efficiency of the parameters displayed on the axes.

In order to further provide a better and faster understanding of the data, the points displayed in the 2D viewport are color coded with regards to the impact they have on the strategy's efficiency. Points which have a greater score are displayed in bright green and points on the graph which have a bad score are represented with red. In this way, the user can quickly scan the 2D space and find concentration points of green color indicating that is an area of convergence for the optimum parameter values.

As position is represented by the x and y coordinates on the grid map, it is not wise to separate these two parameters when displaying them in the viewports. The total number of possible combinations with 2 parameters on the axes and one on the slider for our model of tower defenses is 30, but most of them represent data that is not useful for the user as he cannot quickly reason about it. The identified useful viewports from which the user can choose the path of convergence are: a position viewport with the turret's x and y on the axes and the range parameter on the slider; and 3 viewports that cycle through combinations of turret damage, firing-speed and range on the axes and slider. With these 4 viewports, the user can view the turret parameters from different perspectives and using the coloring scheme can choose the best combination in order to create the local optimum turret.

Local Optimization and Global Strategy

By using the visual analysis tool at each stage (wave) of the game, the user ensures that he will reach the end of the game and have all the necessary information to construct a global strategy. We say global strategy and not global optimum because we want the game designers to have the possibility of generating strategies of different difficulties in order to aid the calibration of the difficulty levels in their games. This global strategy is driven from the choices made at the wave level, where the algorithm points out the optimal choices for that particular state, but the user has the possibility of choosing any values for the turret parameters he wants and proceed further with his selection. Thus the choosing all optimal locals will generate a "hard" difficulty setting while making not optimal choices will generate a lower difficulty setting.

Once the end of the game is reached, the application considers each local optimum as the threshold the strategy must pass for each stage. Also knowing the information about all the waves means that the algorithm can search for combinations of turrets that can achieve the thresholds of future waves at earlier stages. This reduces the necessity to build a turret at each wave and just pumps the parameters of the turret from the earlier stages to handle the later ones if this can be covered by the budget limitations. By analyzing the possible combinations to achieve all of the thresholds, the system can output a global strategy specification indicating to the user when to build a new turret and what its specifications should be.

The important factor in the whole process is that the global strategy bases its decisions on the local choices of the user. This allows users to create not only the best global strategy but also mediocre or weak strategies based on the local decisions and this is how game designers can calibrate their game difficulty levels, while player's will opt for the best local options to generate the best possible strategy.

Results

The finished application allows users to simulate in a test bed environment any kind of software which follows the standard tower defense game model. The modeling of specific game entities is done *via* the application's configuration file. Once the game is simulated, the application can be used as both a videogame providing entertainment value to the user, but also as a development or improvement tool.

Testing has been done on a personal tower defense project. The main reason behind this is that we didn't have access to all the data from a popular tower defense game in time or the game for which we had the data also contained a special feature which was out of the scope of this project which couldn't be modeled. The specification of our tower defense game however fits perfectly as it follows the basic model of such a game. It uses a grid like tile-map where the available tiles could be part of the path from the spawn location to the player's safe point or free spaces on which turrets can be built. The game also featured 2 types of turrets and multiple types of enemies.

We organized 2 types of tests in order to get results from both the perspective of a player improving on his strategy and from a developer trying to calibrate his game standpoint.

In the test regarding a player's strategy we first let the user play through our tower defense game as he would normally

do without having access to our analysis software. The end result of this stage was that he was able to beat the game but made some seemingly bad decisions along the way maybe spending more resources that he should have. He also lost a great deal of his life barely making it to the end of the game.

In the next stage we simulated the same tower defense game in the strategy analysis tool and let the player have another go at it. This time, by using the visualization tools for the turret parameters provided by the software, the player managed to defeat the game quicker and with no health loss.

Finally, after the player beats the game with our software, we let the program generate the strategy based on his decisions and played again through the original game following the instructions given in the generated text file.

We took the data from all three runs and measured the differences in terms of strategy efficiency based on the metric defined earlier. The data showed a big difference between the first run where the player didn't use the software and the second run where he could analyze the impact his decisions had on the game. The difference between the second run and the one where we followed the software generated strategy is not so great, but small improvements were spotted leading us to believe that in the case of other games which are harder than our example the extra computations may pay off.

In the second test scenario we assumed the role of a developer trying to configure difficulty levels for his game. We wanted to generate both an easy and hard level for a configuration of four enemy waves. The test started by configuring the existing level design in the test bed. After that in order to generate the hard difficulty we made the best decisions in turret parameters at each stage and wrote down how well the computer units did at each stage. After the end of this stage we compensated for the difference in the unit's current configuration and the damage output the ideal turrets could achieve. We ran the test another time and analyzed how well the turrets did against the new waves of enemies. After a small number of iterations we achieved a wave composition that could be fully destroyed only if the player made the best decisions in turret parameters and considered this the hard difficulty setting. Further, starting from this setting we ran some more iterations dimming down the specifications on the enemy wave compositions. This allows the user to make turret choices that are not ideal but still survive to the end of the game. With these configurations we achieved the easy difficulty setting.

Both test scenarios proved to be eventually successful which means that the tool can indeed be used for the described purposes. The second test however took more iterations than expected to achieve the desirable difficulty level so further improvement on this aspect should be done.

UNITY GAME ENGINE BASED DEVELOPMENT

Building a videogame or related software from scratch is a long and difficult process which requires a lot of pre-development for the tools and low level software to support it. The game engine chosen for this project is Unity3D because it is a mature software package and has a very active community providing great support [6] and tutorials [7]. Unity also supports coding in C# which is a high-level object oriented programming (OOP) language. This makes it easy to generate your own functionality and modify via scripting pre-existing assets.

Unity deals with objects as entities with a transform component to which multiple other components can be added. The transform specifies the object's location in 3D space, its rotation and scale. Unity treats everything as a component allowing for great modularity and plug and play behavior. Everything from mesh renderers to materials to code scripts are added to objects as components.

CONCLUSIONS

Visual Analysis based Strategy Development

To generate the strategy for the configured game you must first beat all the waves using the normal game mode. Once you finish finding out the individual strategy for each wave you will be presented with a game won screen which has a button labeled "Generate Strategy". If you ask for generating the strategy the system will start displaying a loading bar and begin computation of the strategy based on the choices you have made during normal gameplay. When the loading bar completes a message saying "The strategy has been generated!" will appear. From this point you may choose to close the application and open the text file strategy.txt found in the root directory of the application. The file will contain step by step instructions on how to execute the generated strategy for the tower defense configuration entered in the config.txt file. The strategy text file contains a clear specification of every parameter for every turret needed to execute the tactic and timing instructions as to when you may be able to build each turret. For each wave information about the total budget a player has at his disposal is presented in order to provide a checking mechanism while implementing the strategy. The steps presented in the file may be applied to any tower defense game which fits the configuration specification. This means it is applicable to both another run in the test bed application but also to released games.

Specific Future Development

The important thing to keep in mind is that the current state of the software and algorithms do not produce a **global optimum** solution for any tower defense game even if the user chooses only optimums for each of the waves of the game. The **global strategy** algorithm provides just a small optimization for the choices the player makes at each local stage in order to put locals into the global context. This is by no means the ideal solution. Future improvements of the

tool may include algorithms for devising a true optimal strategy and this could be done similarly as the local optimums through a visualization tool that will help the user search for it in the problem space.

Missing currently from the tower defense strategy analysis application is a dedicated user interface to modify the configuration file for the game. This makes it that each time you want to test bed a new tower defense specification you must enter its details manually in the configuration file and also make sure you respect the format of the file. From own experience this is an easy way to introduce bad data in the game and spend large amounts of time figuring out what went wrong. A special load and save mechanism would also improve the user experience as it would allow storing and sharing of different game configurations at a time.

Another feature that would come as a great improvement on the gameplay and usage of the software is the ability to replay the last wave. Often, mostly when trying to figure out difficulty levels for the game, the user would want to test different turret configurations. In the current setup the player would have to reach the same wave he wants to analyze from the beginning of the game if he wants to test out another configuration. This means reproducing the exact steps that led to that wave. A simple replay wave button would load the game state from the beginning of the previous wave. To support this feature a copy of the game state would have to be saved at the beginning of each wave, and a mechanism for restoring that game configuration when the user asks for replay wave, must be developed.

Finally, for the specific improvements the ability to interact with the global strategy algorithms would benefit any type of user greatly. Currently, the system runs the strategy analysis algorithm behind the scenes and generates what it considers the optimal approach based on the gameplay. However, the computer must go through a lot of data and a visualization scheme similar to the one for local optimums would also let the user guide in the search for global strategies. Another advantage into developing this branch of the application is the fact that a user might be able to generate multiple global strategies based on the previous gameplay.

Currently, the system choses one strategy which is based on calculations, but by performing this task the computer actually generates multiple feasible strategies that are simply dumped. Having access to all the generated strategies the computer develops may help game designers in the process of creating varied game experiences. This feature could be implemented by creating a secondary visualization module for the set of global strategy algorithms.

General Future Development

One direction the project may take includes the expansion of the application to provide its users with reverse engineering tools for game designers. This feature is more suitable for the development of games than for the normal player. It implies that by using the strategy profiling algorithms implemented in the software, a user could generate through the tool a specification for the design of a level in the game.

Another direction for further development is the creation of adaptive AI systems that use the strategy analysis module to change the way the computer plays the game based on decisions made by the human player. For example, on a hard difficulty setting in a game, the AI system could analyze what the player is doing through the strategy analysis module and adapt its units to be stronger or on the limit of the player's build. This way the computer can always change the way the game works offering strong gameplay and great chance for game replay-ability as the experience is never the same.

ACKNOWLEDGMENTS

The research has been carried out in the Computer Graphics and Interactive Systems Laboratory of the Computer Science Department, in the Technical University of Cluj-Napoca.

REFERENCES

1. Gregory Jason, *Game Engine Architecture*, CRC Press 2014, ISBN: 978-1-4665-6001-7.
2. Fraser James, Michael Katchabaw, and Robert E. Mercer, *A methodological approach to identifying and quantifying video game difficulty factors*, Entertainment Computing 5.4 (2014): 441-449.
3. Hsu Shang Hwa, Ming-Hui Wen, and Muh-Cherng Wu, *Exploring design features for enhancing players' challenge in strategy games*, CyberPsychology & Behavior 10.3 (2007): 393-397.
4. Alexander Justin T., John Sear, and Andreas Oikonomou, *An investigation of the effects of game difficulty on player enjoyment*, Entertainment Computing4.1 (2013): 53-62.
5. Aponte Maria-Virginia, Guillaume Levieux, and Stephane Natkin, *Measuring the level of difficulty in single player video games*, Entertainment Computing2.4 (2011): 205-213
6. Unity3D User Manual, [Online]: http://docs.unity3d.com/Manual/index.html
7. Unity3D Tutorials, [Online]: http://unity3d.com/learn/tutorials/modules
8. Uneal Engine official website, [Online]: https://www.unrealengine.com
9. RTS Games, [Online]: https://en.wikipedia.org/wiki/Real-time_strategy