UNIVERSITY OF AGDER

Maximizing Fun in An Adaptive Tower Defense Game using Stochastic Hill Climbing

By

Jon Vegard Jansen, Robin Tollisen

Supervisor: Sondre Glimsdal

Project report for IKT411 in Spring 2013

Faculty of Engineering and Science
University of Agder
Grimstad, 7th of June

Status: Final

Abstract:
Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

Table of contents

"Husk class='notoc'"

Table of figures

"Husk class='notoc'"

Table of tables

"Husk class='notoc'"

1 Introduction

Today, there exists more commercial video and computer games (henceforth: games) than ever before, and the number of persons owning a smart-phone or personal computer is increasing. Many students and pupils neglect their studies in favor of watching television or playing games, maybe because they consider it more fun. We believe that through the use of E-learning and gamification, which enhances traditional learning methods with elements from games, it is possible to make any learning process more fun, resulting in more interested students and better grades.

As each student is unique, we believe that they also have different learning methods that work best for them, which is why we have chosen to look into adaptive games. In this project we research whether a game can be made, such that it adapts to each player on the fly.

1.1 Background

A lot of research has already been done on the area of E-learning, and there exists multiple definitions. We have chosen to use the following one, because it emphasizes the individual student, which goes hand in hand with individual adaption of application.

"We will call e-Learning all forms of electronic supported learning and teaching, which ... aim to effect the construction of knowledge with reference to individual experience, practive and knowledge of the learner..." [1].

Gamification is a concept that has been around for a long time, and has increased with popularity since 2010. [2] [3]. We present the two following definitions, as we think both of them show that gamification can be used well in conjunction with E-learning.

"The process of game-thinking and game mechanics to engage users and solve problems." [3].

"A process of enhancing a service with affordances for gameful experiences in order to support user's overall value creation." [4].

The latter definition is directed towards service marketing, but if one reads 'service' as 'learning process' and 'user's overall value creation' as 'learning outcomes', we believe that gamification is something that could be used together with both tradition learning methods and E-learning.

While we believe in the concept of gamification, for this project we will create our own game to test an algorithm for adaptive games.

If one is to create a game that is meant for learning purposes, it is important to note that the game needs to be fun. [3]. If a game is not fun, it will not be able to educate either, because players lose interest in games the same way as students lose interest lectures, should they be boring. This is why it is important to figure out how it is possible to maximize the fun in any kind of game, thus keeping the interest of players. Since games appeal to a wide variety of players, the perceived level of fun may often vary among its players, due to personal preferences. In order to accomodote for such varying preferences, traditional games have often implemented several levels of difficulty the player may choose from, as well as providing the player with several settings or options for how they want the game to behave. When it comes to difficulty, other work has already been done on how to scale and adapt this to the level of player, in order to make even games, but in order to make a game fun, we believe that more than difficulty needs to adapt to individual preferences, thus more aspects of a game should be able to change on the fly. [5].

1.2 Problem Statement

Our project is to research whether it is possible to create a game, that uses an unsupervised online learning algorithm, which will adapt the game to the user's preferences based on limited user feedback.

1.3 Problem Solution

We have chosen to create a game using the tower defense genre, as this requires somewhat less work and graphics than many other genres, as well as having a lot of possible parameters that can be tweaked in order to adapt the contents of the game (henceforth: gameplay). The game uses a modified stochastic hill climbing algorithm together with some user feedback after each game, to adapt itself towards the individual player. We call the game Adaptive Tower Defense, and abbrieviate it AdaptiveTD.

1.4 Report Outline

Quick and short and concrete

2 Adaptive Tower Defense

This chapter gives some pictures and a short description of our game, as well as some general information about the game genre, namely tower defense.

Figure 1: Left: AdaptiveTD. Right: A tower defense game called Fieldrunners [6]

2.1 A short overview

AdaptiveTD is a real-time strategy game, more precisely it would be referred to as a tower defense game. As you can see in *Figure 1*, both images have some sort of path that the enemies, like the ones below in *Figure 2*, can walk upon.
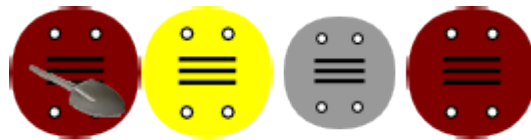

Figure 2: Enemies from AdaptiveTD

The goal of both games is to build enough towers, like the ones in *Figure 3* below, that will try to kill the enemies, before they can cross the path. If too many enemies cross the path, the player will lose.
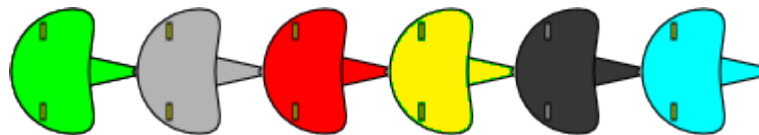
Figure 3: Towers from AdaptiveTD

In order to build towers, the player needs money/gold represented by the number next to the symbol seen in *Figure 4* below. Whenever the player to kill an enemy, it will bring in gold to his or her money bank.

Figure 4: Gold is represented by the number next to this symbol

Already here, there are a lot of parameters that an algorithm can tweak in order to maximize fun. For instance, one could change the balance between the build cost for each tower, and the amount of gold each enemy killed will yield. Another example is to change the health of enemies, making them more or less vulnerable to attack, while at the same time adjusting the damage that each tower does, so that the overall difficulty balance is not changed. In Parameters and their importance we provide a detailed list of many of the chosen parameters, how they impact the game, and why they were chosen.

We created AdaptiveTD using LibGDX, which is "a cross-platform game development library written in Java", that "abstracts away the differences between writting desktop, Android, iOS and HTML5 games based on standards like OpenGL ES/WebGL", where "Applications can be prototyped and developed entirely on the desktop, then only 6 lines of code are needed to run your app on Android or HTML5." [7].

This makes it possible to test and possibly publish AdaptiveTD on Android and iPhones, as well as tablets and computers, both for desktop in all major OS's, and for HTML.
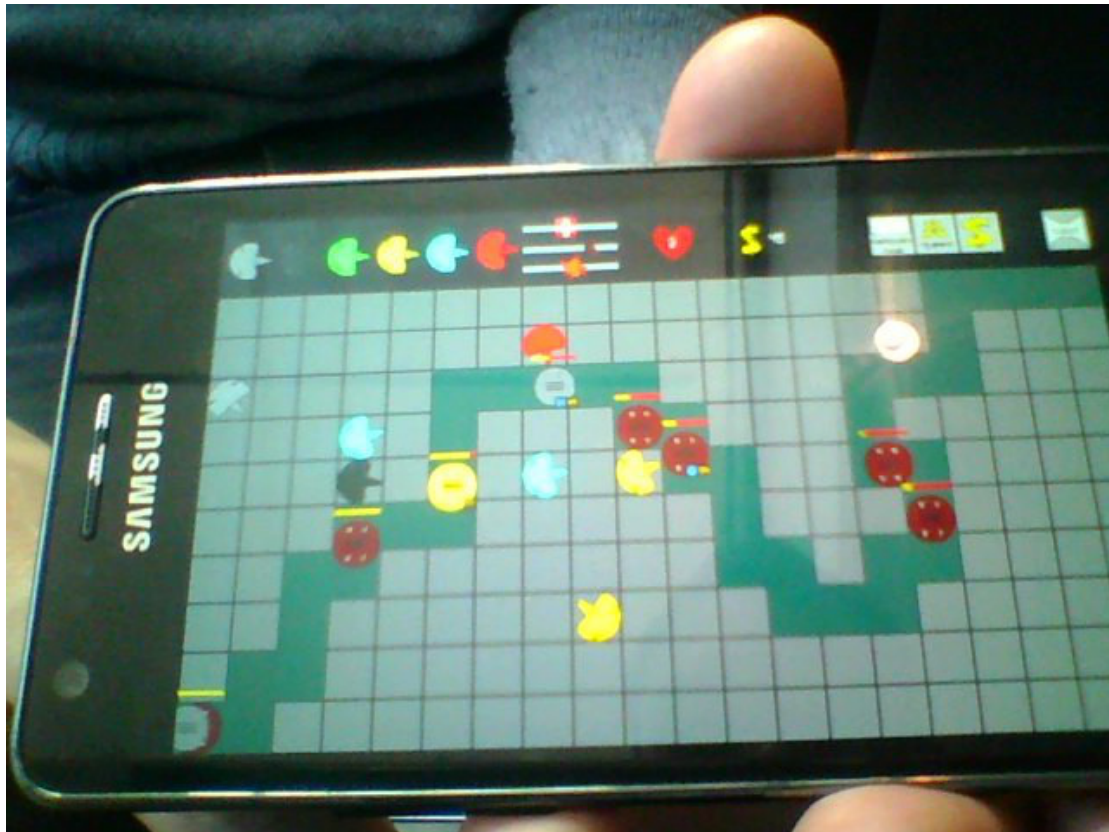
Figure 5: AdaptiveTD on a mobile device

2.2 Parameters and their importance

In the following sub-chapter we provide some of the parameters we keep track of in AdaptiveTD, what they represent, and some general comments on their usefulness.

2.3 AdaptiveTD vs. similar games

AdaptiveTD is very similar to classic tower defense games, in that the player has to stop different kinds of enemies from reaching the destination on their path, by using different kinds of towers. AdaptiveTD differs slightly from other tower defense games on several points.

There are no scripted or static levels, since each game is a step closer towards the player's 'ideal game', that is, the game that is perceived as the most fun. Other than that the game learns after each level, the player has nothing that is carried over to the next game, such as tower access, special items or game progress in general. The player simply plays successive games until he or she quits.

The game has modular towers and enemies. This is not necessarily new to the tower defense genre, as not all tower defense games may have pre-defined enemies and towers, but this is something that can be used in order to adapt the game towards personal preferences. (Guided) Randomly generated maps. Not necessarily new to the genre either, but is another useful technology that can be used to adapt the game to the player. Note that our game did not use this for game adaption, only to make the game seem less static.

3 Stochastic hill climbing

A hill climbing algorithm is an iterative improvement algorithm, searching for a local maximum. It tries to maximize a function f(X), where X could be a node, state, position or, in our case, a vector of parameters. It compares neighboring nodes to the one it is standing on, and checks if they are an improvement on f(X). If one of the other nodes is improving the situation, then the algorithm moves to the new node, commonly the best node, and repeats the process. The process is repeated until no more improvements can be found, thus the algorithm has found a local maximum. [8].

@figure(pseudoshc, pseudoStochastic.png, Pseudo-code for Stochastic Hillclimbing, as done in Stochastic hill climbing example.)
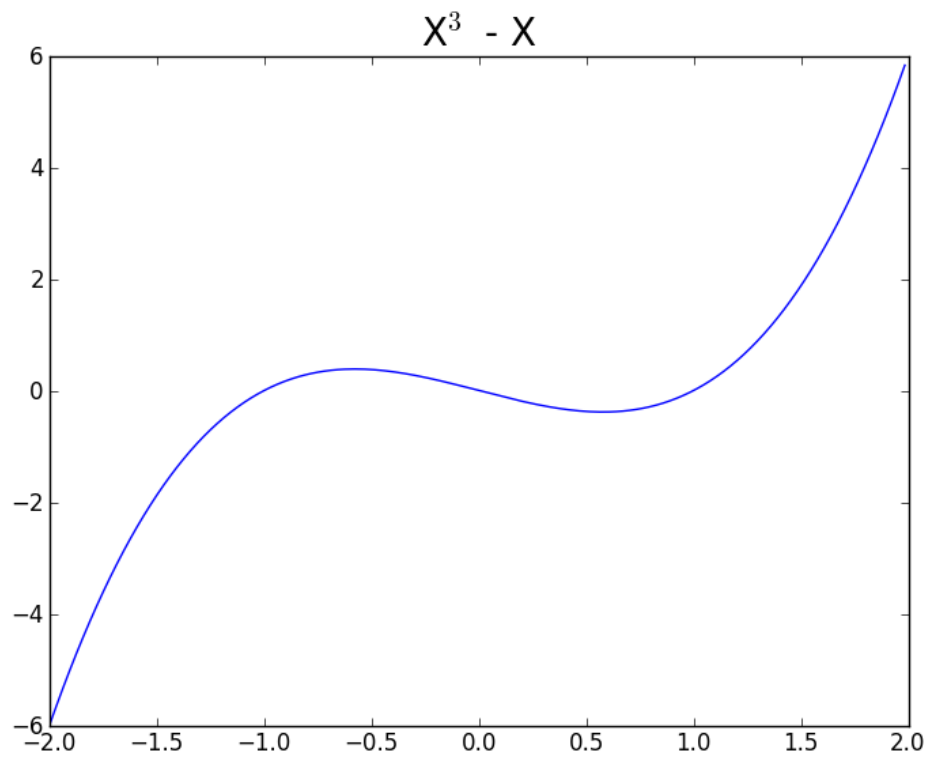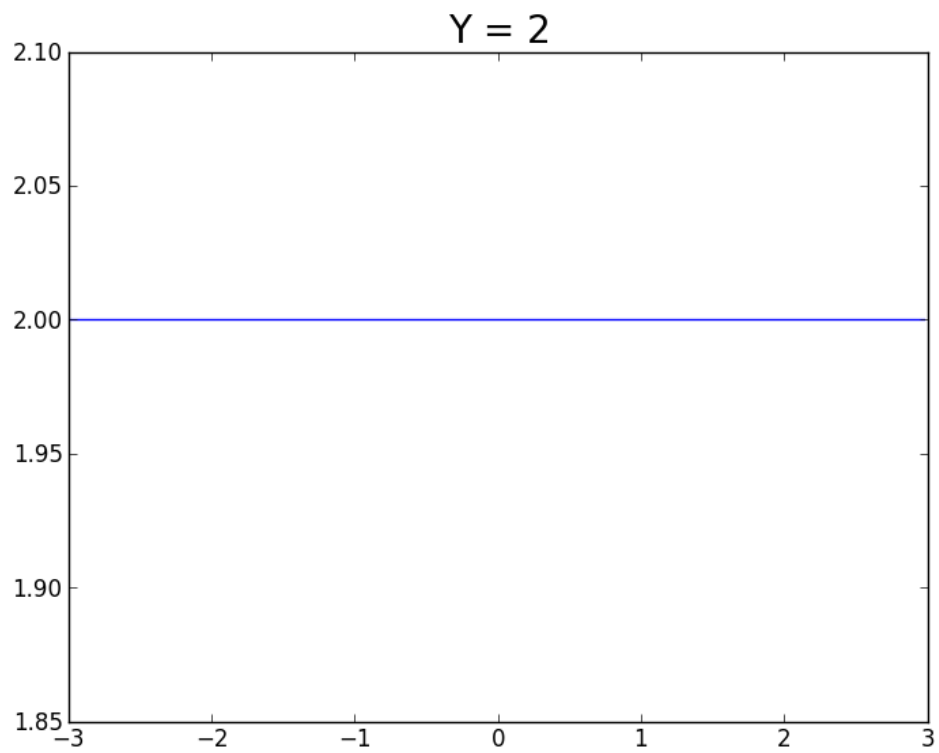

Figure 6: A function that has a local maximum

Figure 7: A function that is too flat to search on
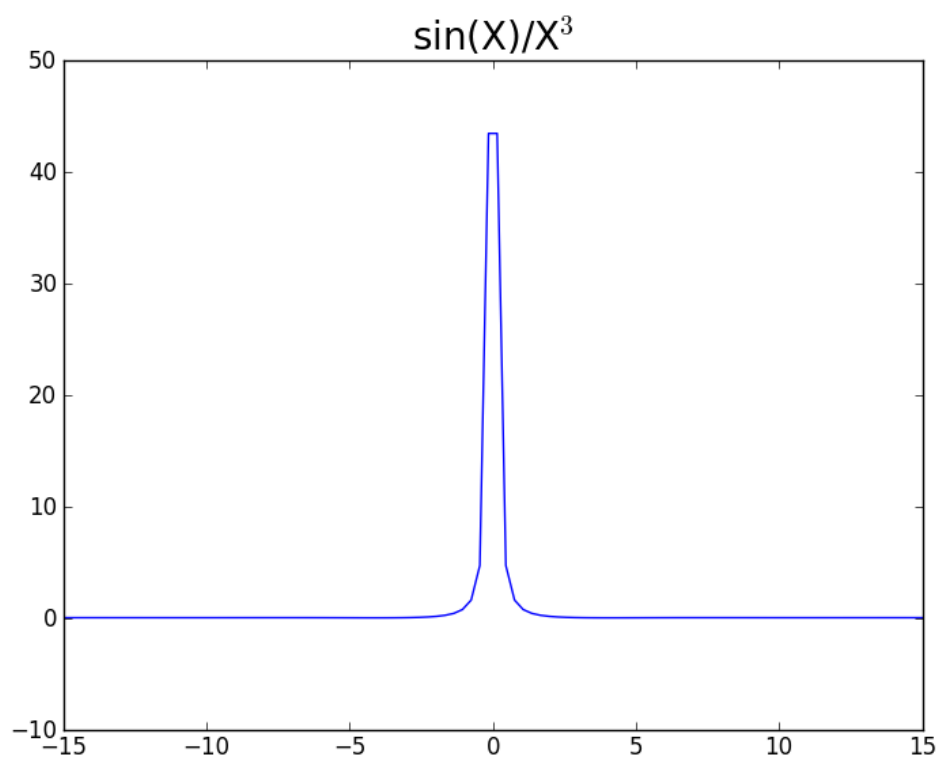
Figure 8: A function that is very steep

One issue with this algorithm, is that it can become stuck on a local maximum, which may not be the global maximum, as in *Figure 6*. One way to reduce the probability of this to happen, is to allow the algorithm to jump larger distances each jump, or to make sure that the landscape is convex. Another issue shown by *Figure 7* is that the search area may be flat, that is, each node is as good as all its neighbors. In this case, any jump will not bring any improvement. A third issue in *Figure 8* is that a node may be lying on a very steep ridge, that is, the 'area' of parameters that give a good node is very small, and can thus be hard to find. [8]

There are different variants of hill climbing algorithms, with different ways of deciding where and when to jump. The stochastic hill climbing algorithm, closely resembling simulated annealing, chooses a position at random, then evaluates whether it is an improvement, and if it is, jumps to the new position. If the new position is not an improvement, it will go back to the last one. With stochastic hill climbing, it may be wise to set a specific maximum jump distance, and let this decrease over time. This allows the algorithm to converge or settle on a position after some time. [8]

3.1 Stochastic hill climbing example

Here is an example for how to use stochastic hill climbing for how to find the maximum value of the function $f(X) = 2 - X^2$. which has a maximum value of 2. We initialize the maximum jump distance (in x-direction) to a value of 1.0, and let this decrease by 0.1 with each jump.
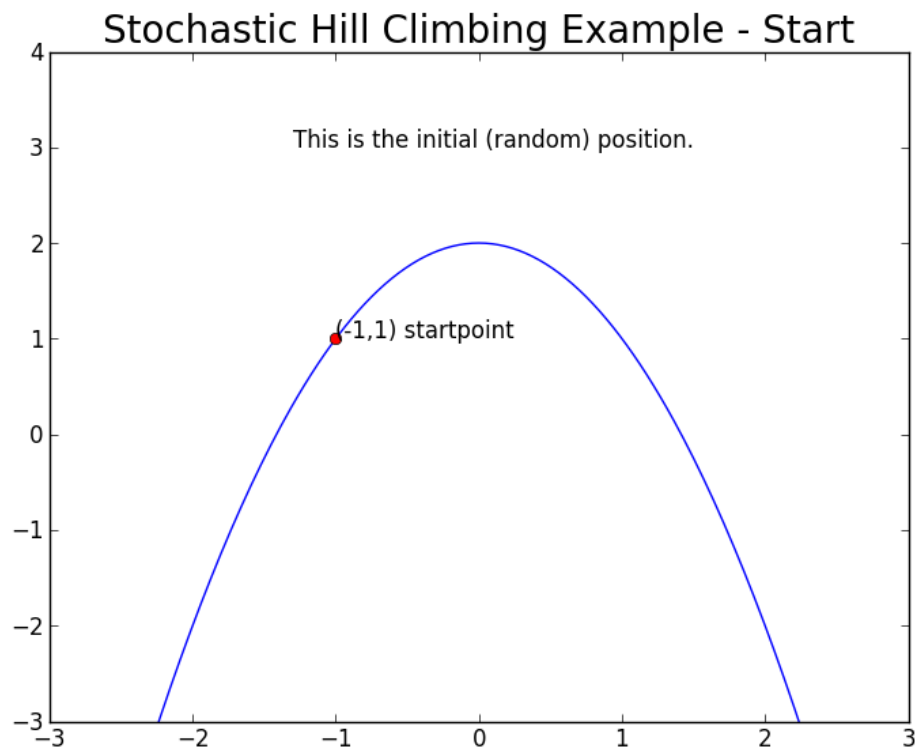
Figure 9: The startpoint for the search
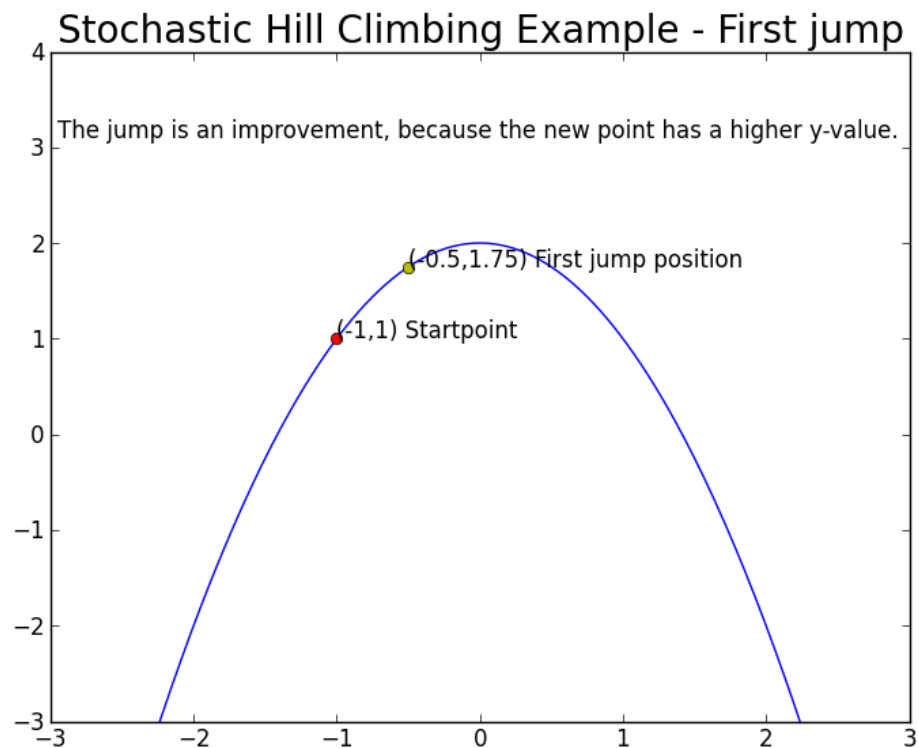
Figure 10: The first jump
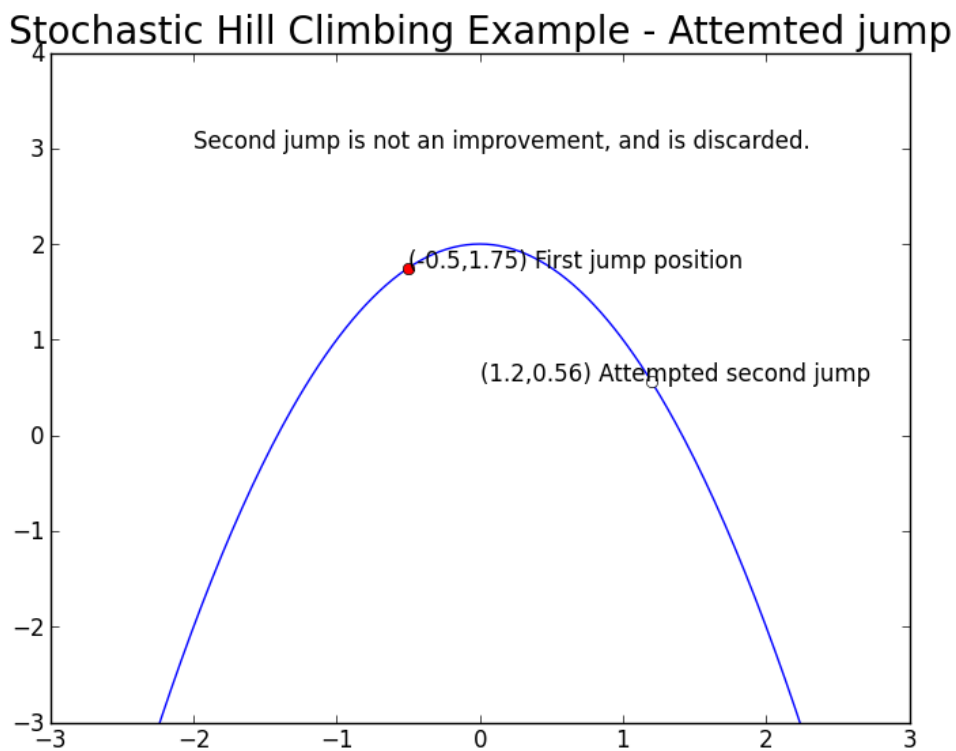
Figure 11: An attempted jump

Stochastic Hill Climbing Example - Global Maximum

This jump achieves the global maximum.

(0,2) Global maximum

(-0.5,1.75) Previous position
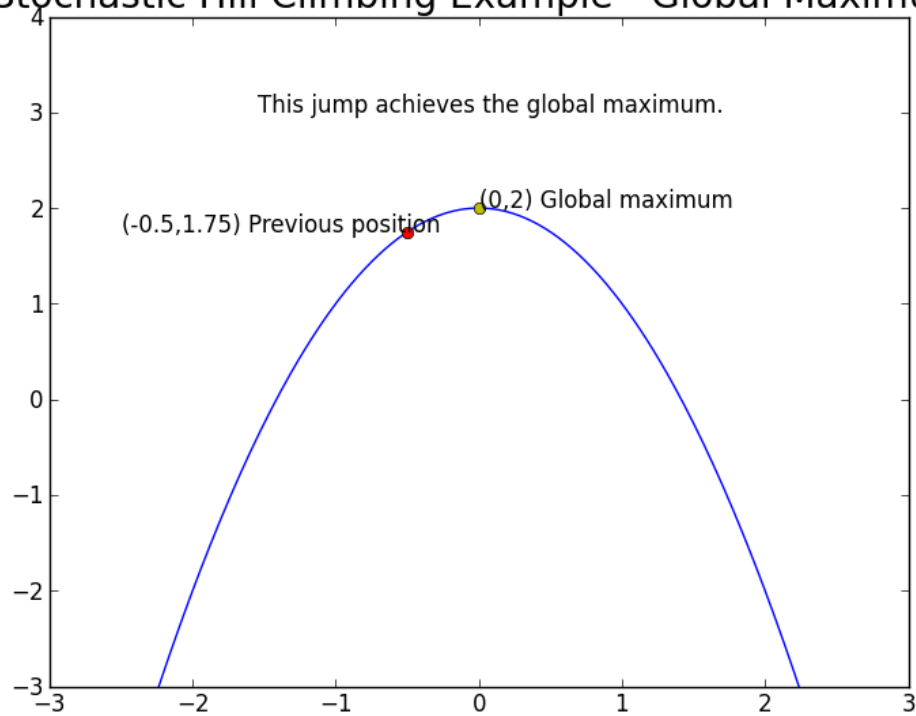
Figure 12: Achieving global maximum

Stochastic Hill Climbing Example - Further Jumps

Further jumps will not be valued better than global maximum, and will be discarded.

(0,2) Global maximum

(1,1) Attempted jump

(-1.3,0.31) Attempted jump
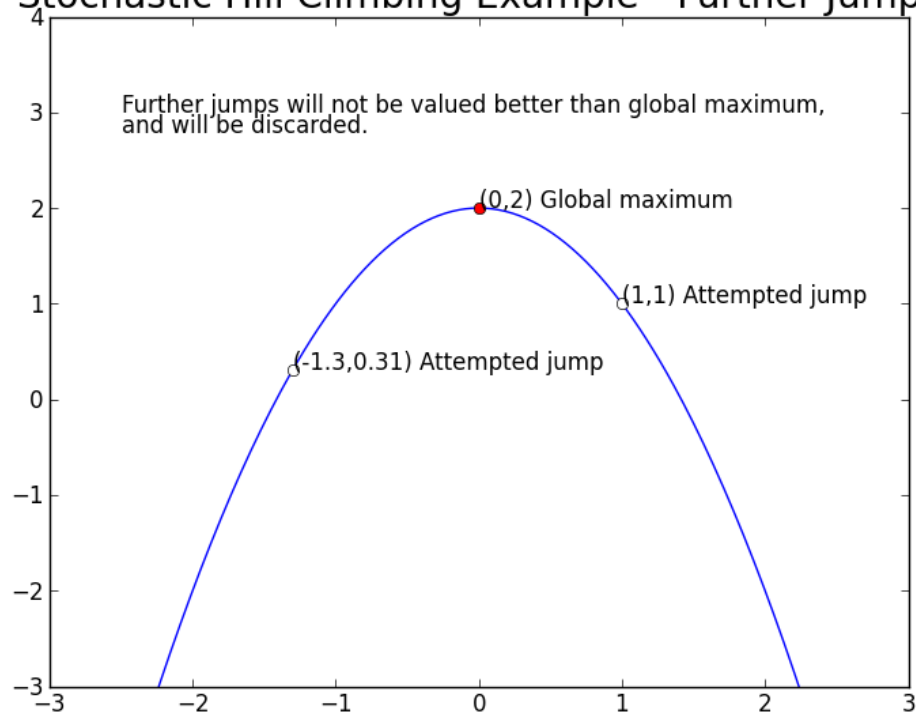
Figure 13: Further jumps are discarded

This is a short example of how stochastic hill climbing will eventually find the local, or global, maximum on a graph. This example is very short, compared to how many jumps it would normally take, but this is actually possible.

With a starting maximum jump distance of 1.0, decreasing by 0.1 with each jump, the algorithm will have converged after ten jumps, hopefully at an acceptable position. In a game setting, it may be wise to let the maximum jump distance never decrease to zero, because it can make the game feel more lively.

4 Solution

In this chapter we will talk about our solution which is an implementation of the Stochastic Hillclimbig algorithm along with a few features needed to adapt it for our use. We work with the paramteres listed in Appendix I - Parameters.

4.1 Algorithm

```
1    Initialize all parameters and relations
2    while playing
3        User plays the game
4        User Gives feedback
5        Create Metric from feedback
6
7        if newMetric > oldBestMetric
8            oldBestMetric = newMetric
9            save parameters as best known parameters
10       else
11       restore best known parameters
12       Jump relations and parameters.
13   end
```

Figure 14: Pseudo-code for the algorithm.

Figure 15: The flow of our algorithm.

As indicated in *Figure 14* and *Figure 15* we start with set values for all parameters, which is a prior we have found to give a decent game. After playing a game the user gives feedback we use to create the metric is "H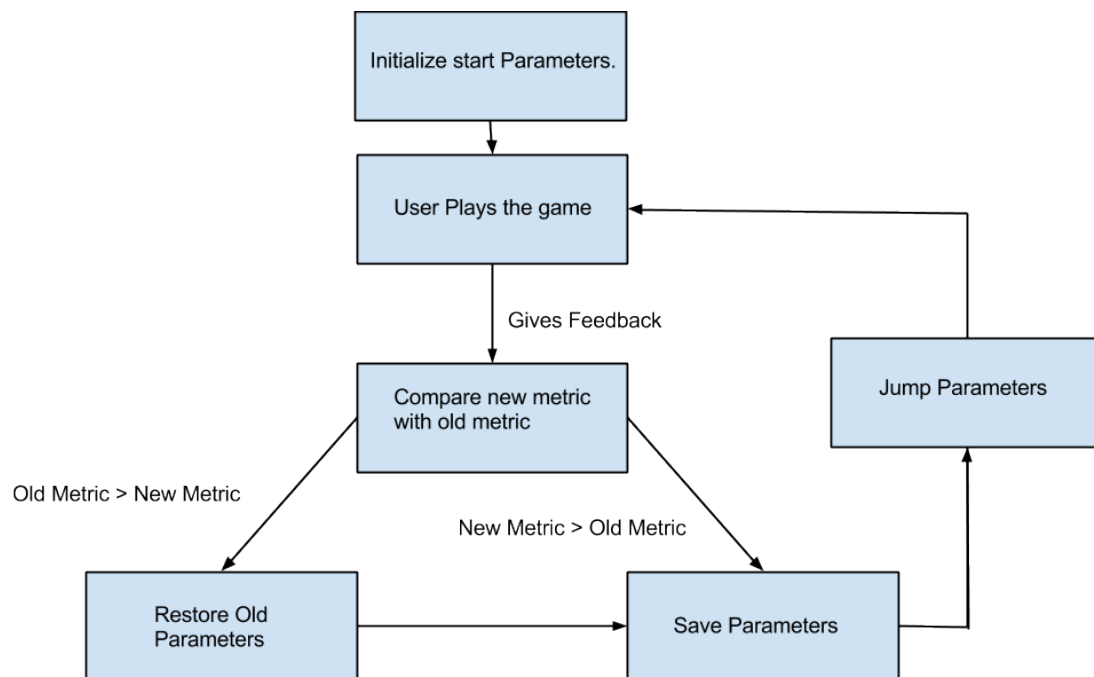ow fun was this game?" and "How difficult was this game?". The first one presents the user with three hearts, and for each heart selected we roll a die from one to ten one time. This number is multiplied with a multiplier which increases as more games are played. This is to not end up stuck in a place just because the feedback received a high random value.

4.2 Litt lang setning, som er en hel paragraf?

After the metric has been created we compare it with the currently best known metric, and if we find that the new metric is better we save these as the best known metric conintue to jump from there, else we restore the parameters from the game where we received the best metric.

```
1   for each relation r with parameters p1 and p2
2           find max change for p1 without overstepping max jump distance
3           if p1.max > maxJumpDistance
4                   p1 += random from -maxJumpDistance to maxJumpDistance
5           else
6                   p1 += random from -p1.max to p1.max
7           end
8           Modify p2 based on the change to p1 as defined by relation.
9   end
```

Figure 16: Pseudo-code for jumping of relations

We randomly increase or decrease the parameters with parameters from zero up to a max jump distance, and let the user play again and repeat the process. The max jump distance decreases each time we find a better game, and thus we attempt to converge at a point where the user enjoys the game. Perceived fun is fleeting though, and if the user grows tired of an area he might start to rate it lower, and thus when a user gives a game the lowest rating we increase the max jump distance, to enable users to find new games.

The main feature we have implemented to adapt the stochastic hillclimbing algorithm to work with our parameters is relations. We needed to change the game, while not necessarily changing the difficulty. To do this we decided to put most of the parameters in relations. We do not change all parameters individually when jumping, but cycle through all relations and jump parameter one, and parameter two is changed as defined by the relation.
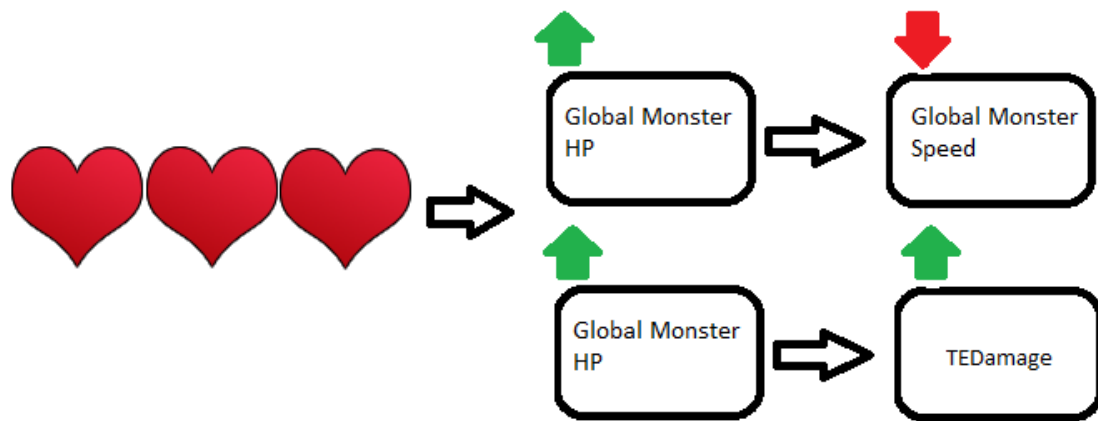
Figure 17: Basic relations example.

The figure above illustrate how relations work. We have the relation GlobelMonsterHP->GlobalMonsterSpeed, when one goes up, the other one goes down proportionally. This may also be between two parameters which will change in the same direction as also seen in *Figure 17* where GlobalMonsterHp->TEDamage(Tower Damage) and both increase. As you may have noted, parameters can also be in more than one relation, thus when GlobalMonsterHP changes, it affects both GlobalMonsterSpeed and TEDamage.
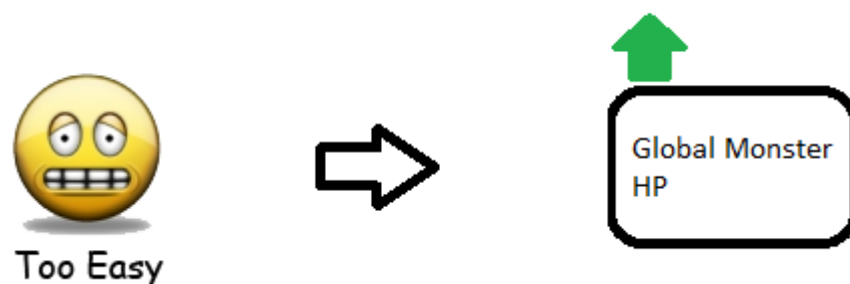
Figure 18: Difficulty does not change in relations.

We also get feedback on the difficulty, while this is not the main focus of this report, it does indicate something about the user. Since a game that is to easy or to hard is not fun, we found that we needed to implement some sort of difficulty scalign as well. We have implemented Stochastic Hillclimbing for the difficulty. Here however we just have parameters, and since the metric here in the sense of a graph would be "Too Low"(Easy), "Perfect" and "Too High"(Difficult), it is quite easy to guide the search and make it less random.

5 Testing

Our testing is split into two parts. The first part contains two case tests of the algorithm, and the second part contains tests from our fellow students and friends.

5.1 Difficulty testing

This is how our algorithm performed.

5.2 Fun testing

After we finished the game, we ran several tests to research the viability of our algorithm.

5.2.1 Real Playtests

We ran some actual playtests, having people who had never played the game before download and play it. These people varied in age, computer background, and experience with games, to get a sense whether we could please a wide variety of people by adapting to their preferences.

While it is hard to plot exactly what they thought, since measuring fun is not an exact science we have gathered some feedback from them which we will summarize here.

- The game is fun to begin with, but gets somewhat monotone when you play it for an extended period of time.
- It is exciting when the game changes radically and new stuff happens that you have to deal with.
- The difficulty scales well, and you can feel the changes quickly if you are dissatisfied with your game.
-

There are also some stats that certain people enjoy whiel others do not enjoy, and since the jumping is currently random for 15 parameters you may end up with not getting everything you desire. The main thing we heard here was that the diggers, as talked about in Parameters and their importance can quickly become overwhelming if you get an unlucky jump while the max jump distance is still high. While you should generally end up with your old parameters if you did not enjoy the game, there are cases where the one roll of the die from one heart can give a metric higher than the old best metric. We have discussed never picking the current parameters the best known if a user rated it with one heart, however we have not had the time to do playtests using this, but believe it would give a better experience.

5.2.2 Case tests

We also did some case tests ourselves, where we decided on a parameter which we wanted to maximize or minimize, and tested if we would get the desired result. We created some methods to make these tests quick, and played. For each test we played 15 games, and gave the game three hearts if it went in the correct direction with our stat, and one if not. We also gave the game the feedback of "Perfect" in the difficulty, such that this part of the game did not alter any stats.

Case Test one: Global Reload Time maximized

For this test we attempted to maximize the Global Reload time. Which means that the parameter GlobalReloadTime should preferably go upwards as we play.

Here are the plot from five playthroughs attempting this.
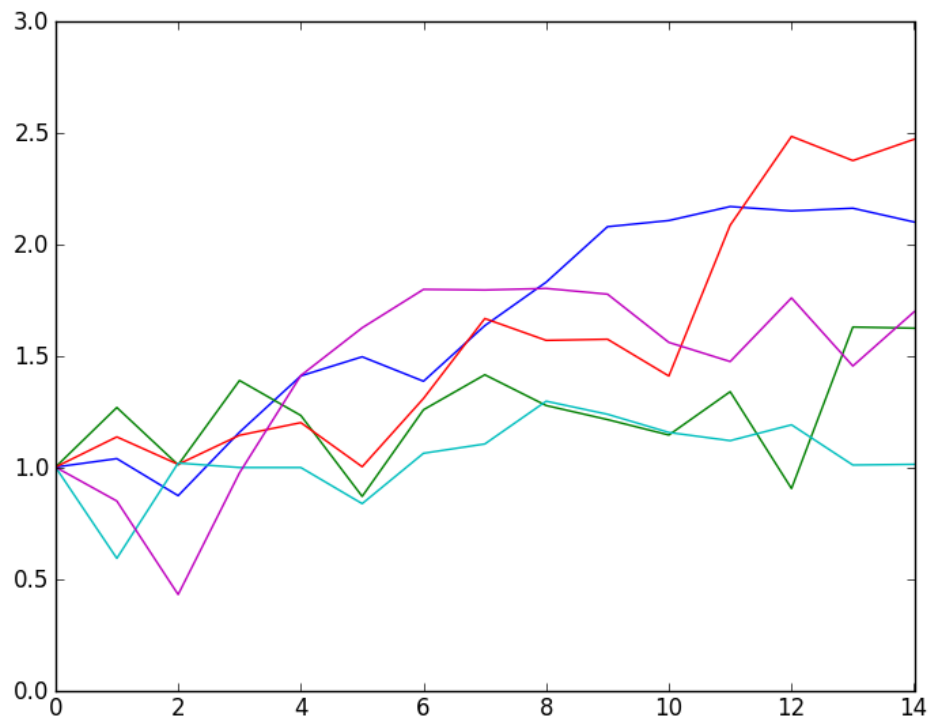
Figure 19: Plot of the case tests for Global Reload time.
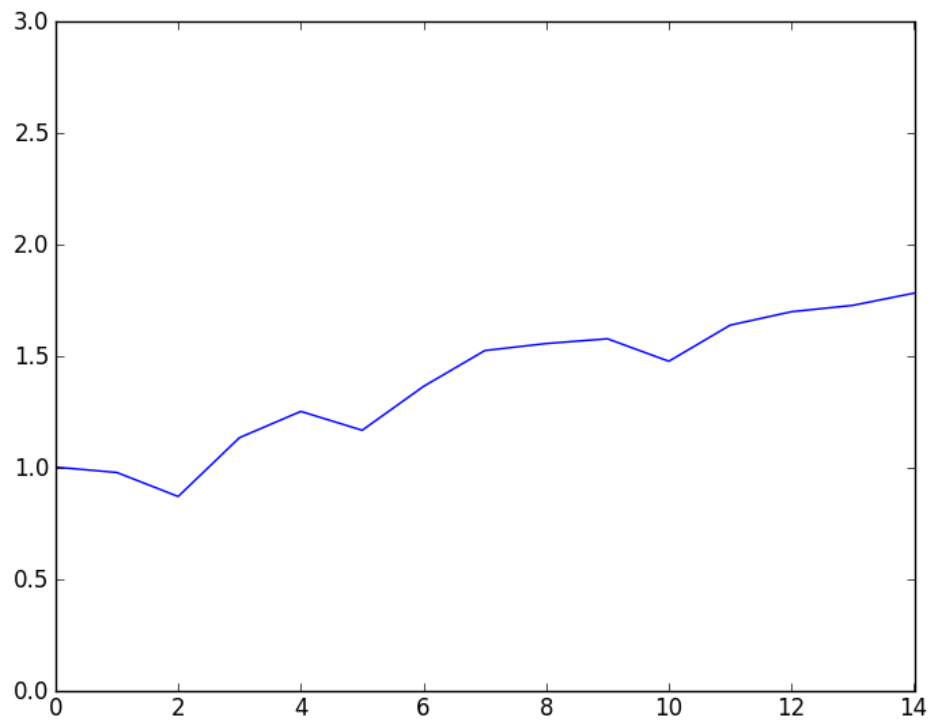
Figure 20: Average values from the case tests for Global Reload time.

As you can see it is moderately successfull each time, but since the jumps are random we do expect jagged graphs as it is as likely to decrease as it is to increase. The trend is however quite clear, the GlobalReloadTime is increasing. We believed that this could also be made even more clear by never choosing the parameters after choosing one heart, as described in the end of Real Playtests after it was pointed out by one of the subjects in the playtest, and we tested with exactly this, as plotted below.
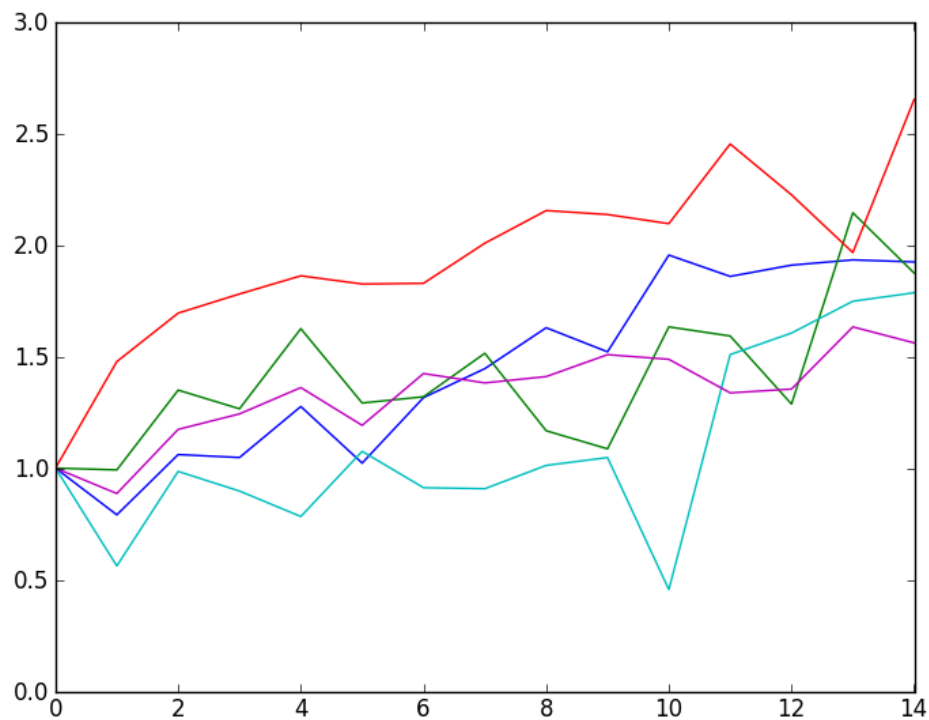
Figure 21: Plot of the case tests after one heart not chosen.

This had little effect, but we do believe that it will help users not get stuck in with a game they disliked as the currently best known game. It performed a little better than the old one on the 15th game, but a 5 * 15 games is not enough to be conclusive. We wish we had the time to automate tests and run the test more times with more games to see whether it is acctually better, worse or equal. Following is the plot of the average from both tests, blue is the average from the tests where one heart could still be chosen as best known, green is the average from the tests where it could not.
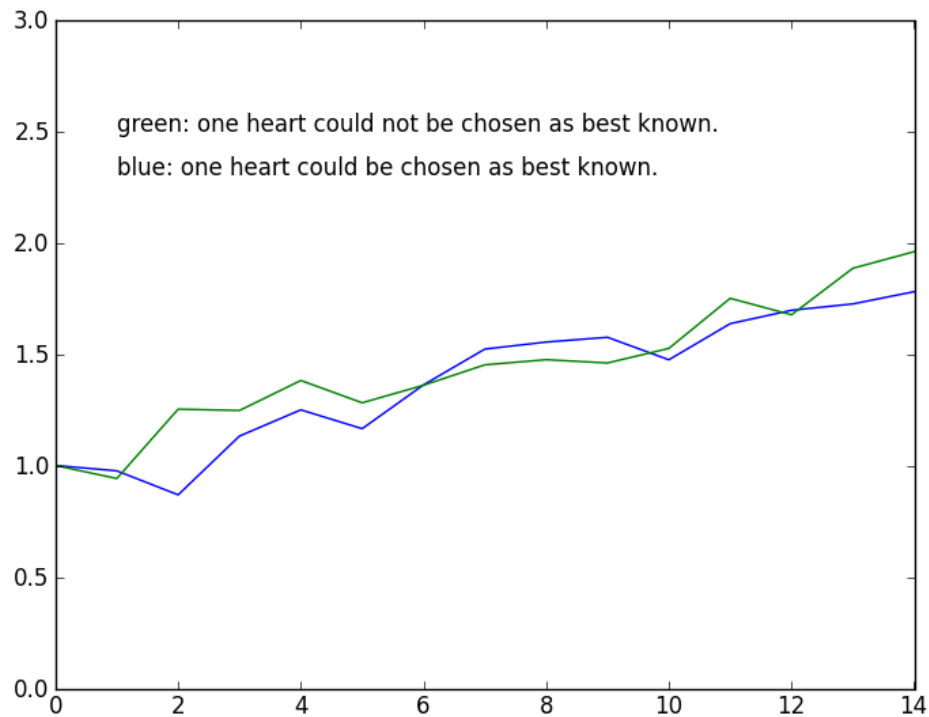
Figure 22: Average Reload time for all playthroughs for both versions.

We also used these tests to see that the relations work correctly. GlobalReloadTime affect TEDamage, such that if the reload time goes up, the damage should also increase. Here is a plot of one of the more successfull playthroughs, with GlobalReloadTime in blue and TEDamage in green. Damage is also affected by other relations, which is why there are deviations in the values.
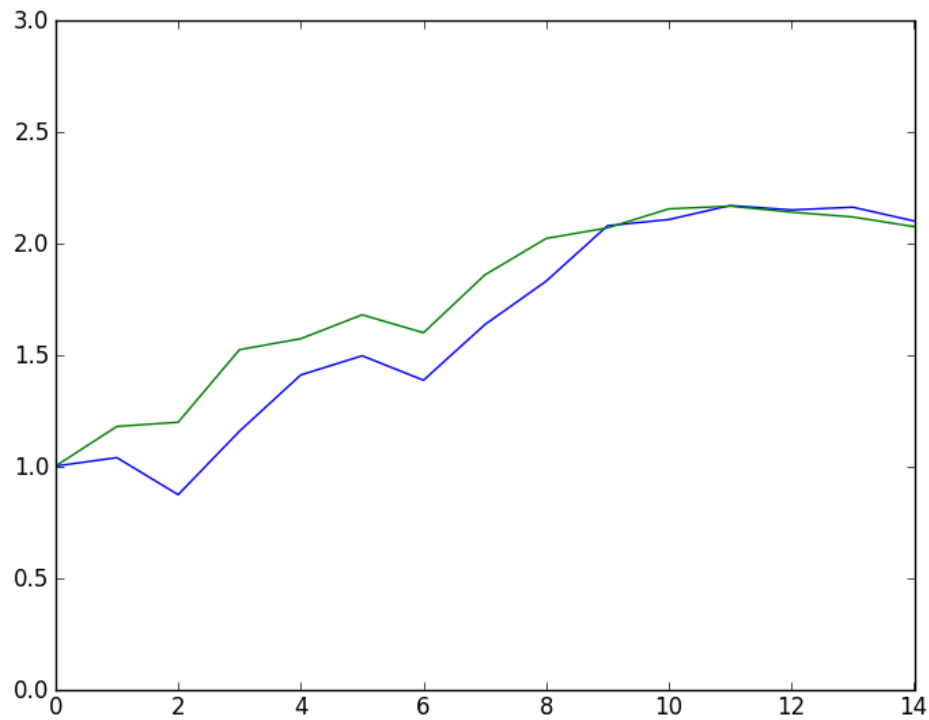
Figure 23: Plot of how global reload and tower damage changed over time.

As you can see they increase almost simultaneously.

6 Discussion

In this chapter, we discuss three topics. First we discuss the results from the fun testing, then the results from the difficulty or case testing, and lastly we discuss some other observations noted during the development process.

6.1 Discussing the fun testing

When looking at the fun testing results, we see that our game was generally rated fun, with an average of 7.17 our of 10. But since our testers were mainly friends and fellow students, we were expecting to receive rather high values, as they did not want to make us feel bad. But a 7.17 out of 10 is not actually bad for the game itself, as much of the game content, such as audio and graphics, was made without a budget. But a difficult aspect of this project, is that it is hard to measure whether something is fun or not. Our main feedback for how well the game worked is the ratings from players, and from that we are satisfied with how fun the game was perceived to be, but even if the game was characterized 10 out of 10, the test results could mostly reflect the game and its contents, and not tell us much about the algorithm itself.

The second part of the fun testing results tells us more about the algorithm itself. The results reveal that the subjects thought the game was actually becoming worse over time, instead of maximizing fun. As mentioned in Fun testing, we believe that two of our parameters, named "SuperChance" and "DiggerChance", see Appendix II - Example Logfile, should have been normalized. As of now, they have too big an impact on the game, because their values are varying too much each jump. To the best of our knowledge, this is a result of our modified algorithm, which we believe can be fixed by having multiple specialized algorithms, instead of a central one. Another solution could be to only alter a single parameter each jump, because the algorithm would then be more sure about what the player enjoyed after each jump. This latter solution would make the algorithm work more slowly, which is maybe a too valuable trade-off.

6.2 Discussing the difficulty testing

The difficulty test results are good, in that the algorithm is able to change the difficulty based on the feedback it receives, relatively quickly. However, the algorithm we use for difficulty scaling is not the same as the one we use for maximizing fun. Since the difficulty can only be either too difficult, perfect or too easy, it is easier to scale difficulty, than to maximize fun.

6.3 Other observations

Another problem we have noticed during the development process, is that the jumps themselves may be too subtle for a player to notice when jump distance shortens. The algorithm requires positive feedback when it goes in a direction the player likes, but if the change is not noticable the player may give it the same score, thus have a decent chance of ending up jumping from his old game each jump. This issue is similar to the one in chapter 5, Stochastic hill climbing, *Figure 7*.

Lastly, we also noticed that the tower defense genre may not be well suited for creating an adaptive game. This is mainly because of the three following reasons. Firstly, the core game elements are rather static, which makes it difficult to make the game look and feel adaptive to the player. A different genre, such as the platformer, with game a example like Super Mario Bros. [9]. Secondly, the parameters in a tower defense game are very tightly knit together, in order to keep the game difficulty in balance. It is difficult to map how they all fit together, which makes it difficult for our algorithm to jump randomly to a new position, while still maintaining mentioned balance. Thirdly, the fun factor in a tower defense game is often related to the feeling of accomplishment. For instance, this feeling may occur when the player has lost a couple of games at the same challenge, and finally is able to beat the game. Since the game adapts itself to the player, thus is never the same, some would feel that their accomplishment is diminished.

7 Conclusion

Although some of the test results came out negative, we still believe that the concept of adapting a game to an individual is something that can be done, in order to maximize fun. The game itself was considered fun, and we believe that some of the reason lays in the variety of the game, caused by its adaptivity. But in order to be able to maximize fun, the algorithm needs to become more refined, and could work much better if we fetch specialized information, indicating what the player really enjoyed. This information could then be used to let the algorithm do a more guided search to find out where to jump next, and not do a completely random jump. On the other hand, the difficulty scaling, works very well, and is something that we can re-use in later projects, such as for E-learning purposes.

As noted in Other observations, the 'flat-issue' that can be experienced with hill-climbing in general, also occurred at some point during our own development process. In order to combat this, an adaptive game itself needs to have much and varied game content, so the different games is felt different by the player. The player can then give better feedback to how the game was, and whether the algorithm should jump from a new location or not.

We would also probably have gotten a better result by choosing a different game genre than tower defense. When the game uses a stochastic hill climbing algorithm relying on too many parameters, it is unlikely that all will go in an optimal direction at once. As mentioned in the second part of Discussing the fun testing, we believe that if the game had used multiple specialized algorithms, which examines fewer parameters at a time, the total algorithm could have been more precise in its jumps. If specialized enough, one of the algorithms could also automatically detect feedback through how a player plays, rather than requiring a questionnaire at the end.

If we do stochastic hill climbing based on metrics created based on how the player is playing, on few parameters for each part of the algorithm, we believe that we could get much better results. However, as this has been, to the best of our knowledge, the first venture into creating games which adapt based on fun, we are quite happy with our result.

SPØRRE ROBIN

8 Future Work

The algorithm works ok, but it needs to be more refined. We believe that with some work refining the algorithm, and doing more testing with different sensors to see if certain feedback can be interprented to liking certain things in the game, we can create a great algorithm to maximize fun even in games with many parameters.

We would like to try to implement this sort of algorithm in a game related to e-learning. Currently one of the problems with e-learning games is that they feel to much like learning and to little like games. If we could be able to create a game which can maximize the fun a player has, while also scaling the difficulty appropriately we believe we could create an effective learning tools for many skills.

With a more refined algorithm we believe this could be highly useful in a game, however it needs more sensors or ways to detect which parameters the players are satisfied with and not. There are also interesting applications for this sort of algorithm in E-learning, if we can detect how to make an e-learning game fun for a player and also scale the difficulty to their ability we may create an effective game for learning.

9 References

IEEE, ACM, Springer

[1] Tavangarian, D., Leypold, M. E., Nölting, K., Röser, M., & Voigt, D. (2004). Is e-learning the Solution for Individual Learning. *Electronic Journal of E-learning*, 2(2), 273-280.
[2] Marczewski, A. (2012). *Gamification: A Simple Introduction.* Andrzej Marczewski.
[3] Zichermann, G., & Cunningham, C. (2011). *Gamification by design: Implementing game mechanics in web and mobile apps.* O'Reilly Media.
[4] Huotari, K., & Hamari, J. (2012, October). Defining gamification: a service marketing perspective. In *Proceeding of the 16th International Academic MindTrek Conference* (pp. 17-22). ACM.
[5] Spronck, P., Sprinkhuizen-Kuyper, I., & Postma, E. (2004). Difficulty scaling of game AI. In *Proceedings of the 5th International Conference on Intelligent Games and Simulation (GAME-ON 2004)* (pp. 33-37).
[6] http://wp.appadvice.com/wp-content/uploads/2010/04/IMG_0005.jpg
[7] https://code.google.com/p/libgdx/
[8] Russell, S. J., & Norvig, P. (1995), *Artificial Intelligence: A Modern Approach.* Upper Saddle River, NJ: Prentice Hall
[9] http://mario.nintendo.com/

Appendix I - Parameters

Following is a list of the parameters the game changes, and their function.

- GlobalMonsterHP - Affects the HP of all enemies.
- TEDotDamage - Affects the damage of each tick for the damage over time tower.
- TEDotTicks - Affects how many ticks of damage the damage of time tower does.
- TESlowPercentage - Affects the amount of slow the frost tower causes to an enemy.
- TESlowDuration - Affects the duration of the slow from the frost tower.
- GlobalReloadTime - Affects reload time for all towers.
- TEDamage - Affects damage for all towers.
- GlobalBuildCost - Affects cost to build each tower.
- GlobalMonsterSpeed - Affects the speed of all enemies.
- GlobalMonsterGoldYield - Affects how much gold each enemy yields.
- GlobalTowerRange - Affects how far towers can shoot.
- DiggerChance - Affects how often an enemy will be a digger.
- SuperChance - Affects how often an enemy will have one or more super effects.
- EarthquakeChance - Affects how often earthquakes are enabled and disabled.
- EarthquakeChanceInGame - Affects how often there is an earthquake when enabled.

Appendix II - Example Logfile

This is an example log file as generated when someone plays the game.

Game number 1 - at May 23, 2013 12:33:20 PM - Game WON
Lives left : 10/10
Gold left : 90/100
Earthquake count : 9
Shots fired : 607

-Tower information---

Towers built : 12
Upgrades bought : 0
Towers sold : 0
Towers destroyed : 0
Earthquake proofs : 0
Arrow towers : 1
Frost towers : 2
Cannon towers : 2
Flame towers : 0
Laser towers : 4
Burning towers : 3

-Metric and jump information---

Hearts feedback : 0
Difficulty feedback : 0
Last metric : 0.0
Challenger metric : 0.0
Max jump distance : 0.4
Player Level : 0.0
Game length multiplier : 1.0

-Enemy information---

Total enemies killed : 45/45
Basic enemies killed : 14/15
Fast enemies killed : 14/15
Tough enemies killed : 17/18
Diggers killed : 0/0
Total supers killed : 0/0
Super fast killed : 0/0
Super tough killed : 0/0
Super shield killed : 0/0
Super invis killed : 0/0

-Parameters---

TEDotDamage: 1.0 Min: 0.1 Max: 3.0
TEDamage: 1.0 Min: 0.01 Max: 10.0
DiggerChance: 0.2 Min: 0.0 Max: 1.0
EarthquakeChance: 0.2 Min: 0.0 Max: 1.0
GlobalMonsterHP: 1.0 Min: 0.1 Max: 3.0
SuperChance: 0.2 Min: 0.0 Max: 1.0
TESlowDuration: 1.0 Min: 0.1 Max: 3.0
EarthquakeChanceInGame: 0.2 Min: 0.1 Max: 0.9
GlobalReloadTime: 1.0 Min: 0.1 Max: 3.0
GlobalTowerRange: 1.0 Min: 0.1 Max: 10.0
GlobalMonsterGoldYield: 1.0 Min: 0.1 Max: 3.0
GlobalMonsterSpeed: 1.0 Min: 0.1 Max: 10.0
TESlowPercentage: 1.0 Min: 0.1 Max: 1.4
GlobalBuildCost: 1.0 Min: 0.1 Max: 3.0
TEDotTicks: 1.0 Min: 0.1 Max: 3.0

Game number 2 - at May 23, 2013 12:34:49 PM - Game LOST
Lives left : 0/10
Gold left : 21/110
Earthquake count : 0
Shots fired : 324

-Tower information---

Towers built : 11
Upgrades bought : 0
Towers sold : 0
Towers destroyed : 3
Earthquake proofs : 0
Arrow towers : 2
Frost towers : 1
Cannon towers : 2
Flame towers : 1
Laser towers : 3
Burning towers : 2

-Metric and jump information---

Hearts feedback : 0
Difficulty feedback : 0
Last metric : 0.0
Challenger metric : 5.0
Max jump distance : 0.25599998
Player Level : 1.0
Game length multiplier : 1.2

Total enemies killed : 22/45
Basic enemies killed : 8/12
Fast enemies killed : 9/21
Tough enemies killed : 5/11
Diggers killed : 0/3
Total supers killed : 0/3
Super fast killed : 0/2
Super tough killed : 0/3
Super shield killed : 0/1
Super invis killed : 0/1

-Parameters---

TEDotDamage: 1.0799435 Min: 0.1 Max: 3.0
TEDamage: 0.95508564 Min: 0.01 Max: 10.0
DiggerChance: 0.4 Min: 0.0 Max: 1.0
EarthquakeChance: 0.3330284 Min: 0.0 Max: 1.0
GlobalMonsterHP: 1.1467683 Min: 0.1 Max: 3.0
SuperChance: 0.4 Min: 0.0 Max: 1.0
TESlowDuration: 1.2593029 Min: 0.1 Max: 3.0
EarthquakeChanceInGame: 0.3 Min: 0.1 Max: 0.9
GlobalReloadTime: 1.1283836 Min: 0.1 Max: 3.0
GlobalTowerRange: 1.1272081 Min: 0.1 Max: 10.0
GlobalMonsterGoldYield: 1.1011007 Min: 0.1 Max: 3.0
GlobalMonsterSpeed: 0.8071418 Min: 0.1 Max: 10.0
TESlowPercentage: 0.74069715 Min: 0.1 Max: 1.4
GlobalBuildCost: 1.1011007 Min: 0.1 Max: 3.0
TEDotTicks: 0.92005646 Min: 0.1 Max: 3.0