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

Keywords: E-learning, unsupervised online learning, stochastic hill climbing, maximize fun

Abstract:
This document presents the research work done in the course IKT411 - Advanced Project. The project has been to research whether it is possible to create a game, that will adapt itself to the user's preferences. In order to do this, we created a proof of concept game, called Adaptive Tower Defense, and tested our solution by running case tests and real playtests. We conclude that it is possible to create an adaptive game, but great care in game design must be taken. If done properly, this is also applicable for the field of E-learning.

Table of contents

Table of figures

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 in 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 the player.[5] However, 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.

To the best of our knowledge there has not been done much research into how to create adaptive game content, which adapts based on the perceived amount of fun the user has.

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. 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

In this report, we first describe the proof of concept game we created. Secondy, we give an introduction to stochastic hill climbing, and then describe the algorithm used for our solution. Thirdly, we describe our tests, and discuss our results. Lastly, we end with some concluding notes and future work.

2 Adaptive Tower Defense

This chapter gives some pictures and a short description of AdaptiveTD, 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.

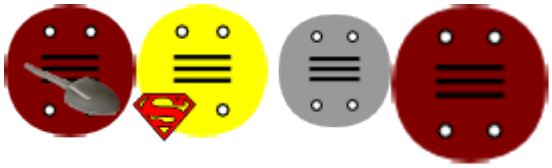

Figure 2: Enemies from AdaptiveTD

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

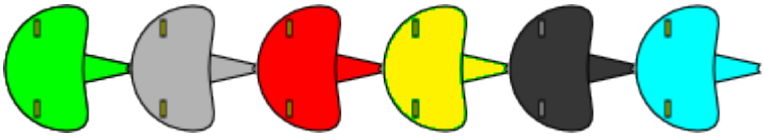

Figure 3: Towers from AdaptiveTD

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 towers kill an enemy, it will bring in gold to the player's money bank.

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

Already here, in the parts of the game mentioned above, there are a lot of parameters that can be tweaked in order to maximize fun. For instance, one could change the balance between the tower build cost and the amount of gold gained from killed enemies. Another example is to change the health of enemies, while at the same time adjusting the damage that each tower does, so that the overall game difficulty is not changed. As you can see, there are a lot of underlying parameters, and small changes can be enough to make or break a fun tower defense game. The problem with all these underlying parameters is that they can be hard to observe, thus the player does not know that the game has actually changed. That is why it is important to give the player some visual feedback when the game changes. An example shown in *Figure 5*, shows that the background color of the game changes in direct relation to the level of enemy health, thus making a more difficult game look darker, and easier games look brighter.

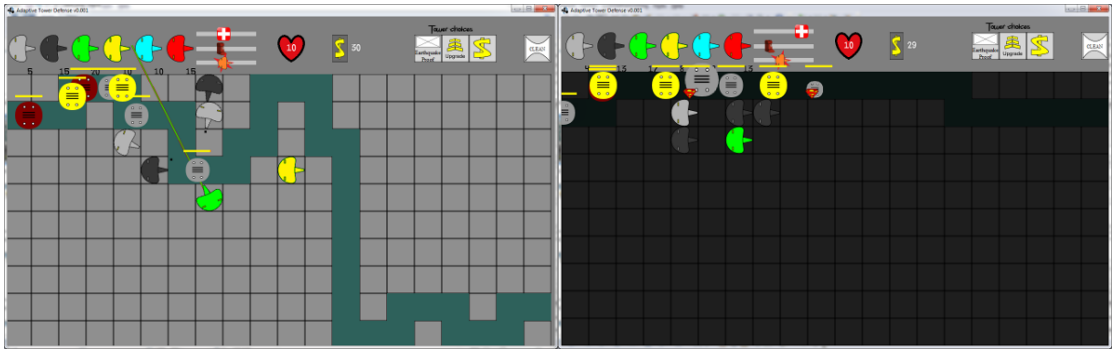

Figure 5: The level of enemy health (difficulty) is reflected in the background color

AdaptiveTD also provides the player with three different sliderbars, see *Figure 6*, different game music playback speeds, based on enemy speed, earthquakes that make the game screen shake and different sizes and symbols on enemies, see *Figure 2*, to inform the player that the game is changing.

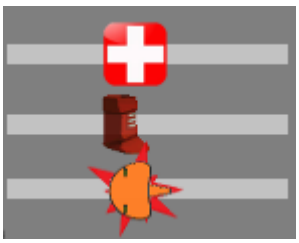

Figure 6: Sliderbars that indicate the current level of enemy health, enemy speed and tower damage

In Parameters and Functionality we provide a detailed list of many of the chosen parameters, how they impact the game, and why they were chosen.

2.2 Parameters and Functionality

AdaptiveTD keeps track of 15 different parameters, that are represented in 12 different relations. Appendix I - Parameters contains a complete list of parameters. In this subchapter we explain the parameters which have the most direct impact on the game. The 12 relations are important to keep the game difficulty balanced at all times, by letting related parameters be adjusted whenever a parameter is changed. In *Figure 7* we provide the most important relations and parameters in AdaptiveTD. Note that the parameters are renamed for explanatory purposes, and appear in Appendix I - Parameters under a different name.

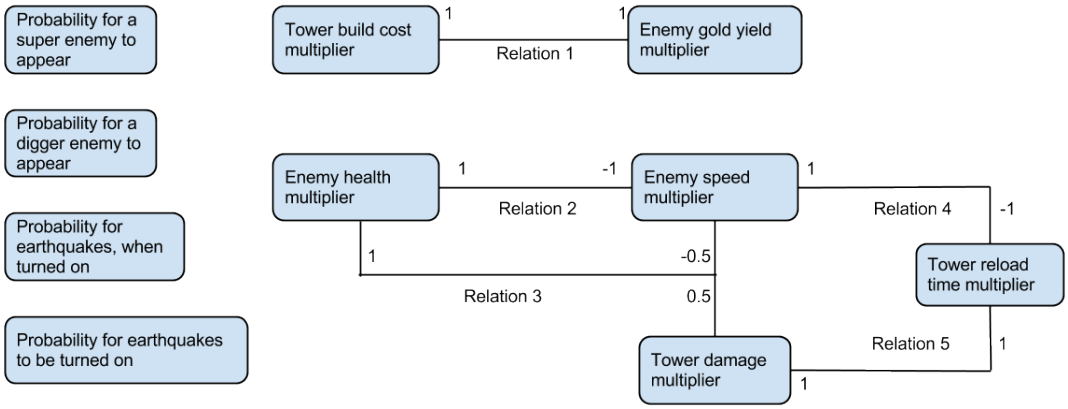

Figure 7: Parameters in relation to each other

Every relation involving at least two parameters has a number next to each parameter. This number is the impact factor, and represents the proportion of how much a parameter should increase or decrease, when a related parameter is changed. For instance in relation 1, Tower Build Cost Multiplier and Enemy Gold Yield Multiplier both have the number '1' next to them. This means, that if one of them is changed by a certain value, the other one should also change with the same value, thus they are still in balance. Relation 2 is inverse proportional, but similar to relation 1, in that if one of them changes with a value, the other one should change with the same, negative value. Relation 3 is different, because it involves three parameters, but it functions in the same way. If Enemy Health Multiplier is changed, then Enemy Speed Multiplier should change by half the negative value, and Tower Damage whould change by half the value. Note that Enemy Speed Multiplier and Enemy Health Multiplier is in the same relation twice. This is not a problem, because if one relation decides to change the internal balance, the other relations are not influenced. So if Enemy Speed Multiplier changes in one relation, the change will NOT propagate to the other relations.

These relations and numbers are what we believe is approximately good enough to keep the difficulty balance constant. For instance, if the enemy health increases by 20%, it should be sufficient that tower damage is also increased by 20%, in order to keep balance. Of course, towers that do not deal direct damage, but have a more tactical effect, such as slowing the speed of an enemy, will be somewhat less efficient after a change like this.

To the left there are four parameters that are not in relation with any other parameters. They are still in relation with themselves, in order to be able to jump, more about jumping in the Algorithm chapter. These four parameters hold the probabilities for digger and super enemies to appear and for earthquakes to occur, all of which are described more in detail below.

Probability for a super enemy to appear is a value from 0 to 1 that the next enemy generated will have some super properties, i.e. have more health, more speed, sometimes be invisible and/or be impossible to harm for a short time period, as seen in *Figure 8*.

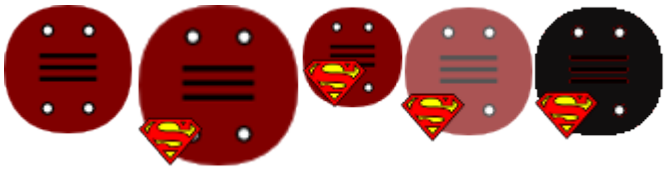

Figure 8: An enemy who has been given extra, super abilities: Normal, more health, more speed, invisibility and impossible to harm

Probability for a digger enemy to appear works a little differently from super abilities. A round of the game consists of 45 enemies, with four possible types of enemies that can be generated. The "Probability for a digger enemy to appear" parameter is simply the probability that each of these 45 enemies is a digger. So a value of 0.67 out of 1, should result in approximately 30 diggers, which is really a challenge. In the Discussion chapter we discuss some weaknesses of this approach. Below is a figure of a digger enemy and what it can do.

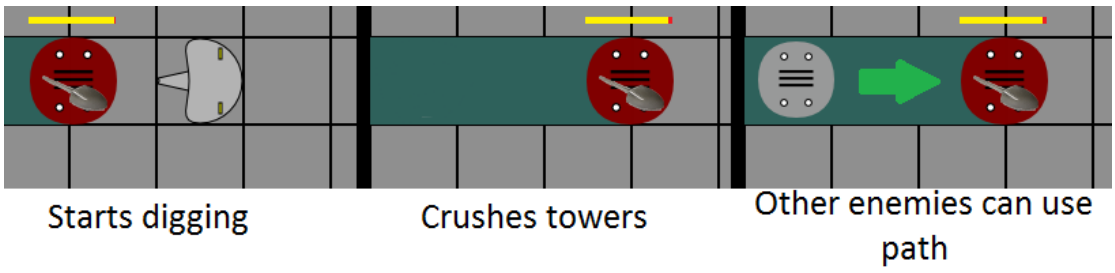

Figure 9: The digger enemy is a difficult opponent

The earthquake functionality shakes the game screen, causing towers to be randomly moved, unless they have been placed in a place where they cannot move or upgraded with an anti earthquake upgrade. The earthquake is dependent upon two parameters, the first which is the Probability for earthquakes to be turned on and off every five seconds, and the probability that earthquakes occur, when they are on at all. In Discussion we discuss the earthquake functionality, as we received some mixed feedback on it.

The rest of the parameters are more subtle, only changing underlying variables. In the Algorithm chapter we go through in detail how the game jumps to change its content and difficulty.

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. AdaptiveTD features modular towers and enemies. This is not necessarily new to the tower defense genre, as not all tower defense games may have predefined enemies and towers, but this is something that can be used in order to adapt the game towards personal preferences. AdaptiveTD also has randomly generated maps. Not necessarily new to the genre either, but is a useful tool that can be used to adapt the game to the player. Note that AdaptiveTD did not use this for game adaption, only to make the game seem less static.

2.4 Tools

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 even in HTML5 with some slight modification.

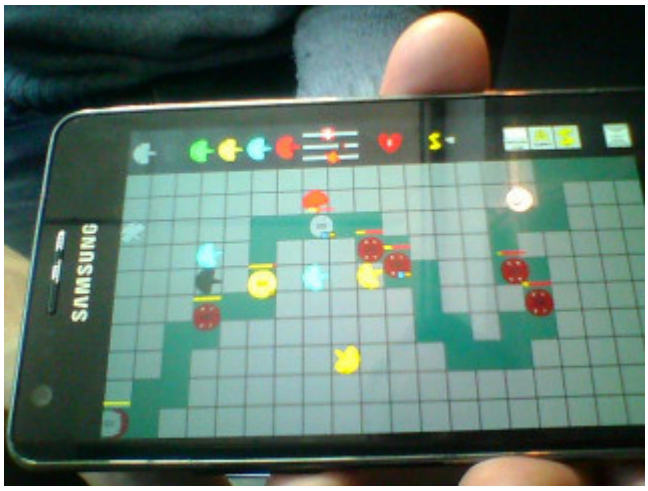

Figure 10: AdaptiveTD on a mobile device

Since this is only a proof of concept game meant for testing purposes, we are not planning on publishing the game, nor make it publicly available. However, in Future Work we present some possibilities for further development and use for AdaptiveTD.

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].

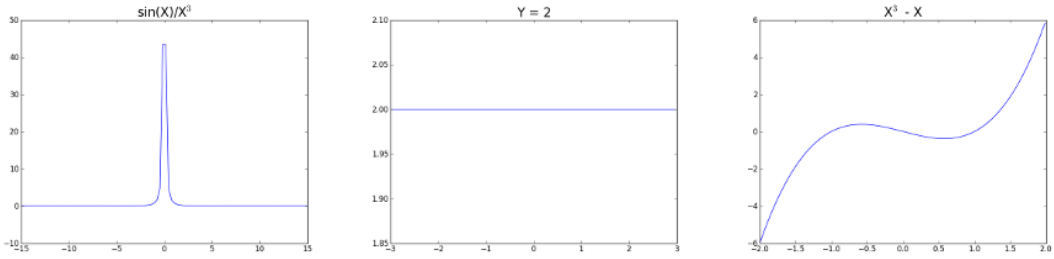

Figure 11: Some issues with Hill Climbing

In *Figure 11* above, there are three issues with this algorithm, that the user must be aware of. One issue is that the algorithm can become stuck on a local maximum, which may not be the global maximum, as to the right. 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. In our case, the player probably has more than one area that he or she considers fun, so when the player is bored by the area he or she currently is in, AdaptiveTD expands the maximum jump distance to try to find other local maxima. Another issue, shown in the middle, is that the search area may be flat, that is, each node is as good as all its neighbors, and any jump will not bring any improvement. A game with little or no visual feedback, would probably have been subject to this issue, but through the of use visual effects, we believe that this effect can be reduced. A third issue, as seen to the left, 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]. Since the preferred area for each individual players is different, this can be a problem for players who enjoy very few variations of AdaptiveTD.

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].

```
f(x)=2-x^2
while do Stochastic Hillclimbing
    choose oldX = random start x
    newX = oldX
    jump newX by random value between -max jump distance and +max jump distance
    if f(newX) > f(oldX)
        oldX = newX
        reduce max jump distance
end
```

Figure 12: Pseudocode for stochastic hill climbing using example in the following subchapter

The pseudocode shown above in *Figure 12* is an example of a search for maximum value, using stochastic hill climbing. Given enough time, stochastic hill climbing should be able to locate the maximum value, however there are some issues with it, that may cause problems.

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 that is making an improvement. The example uses the pseudocode from *Figure 12*.

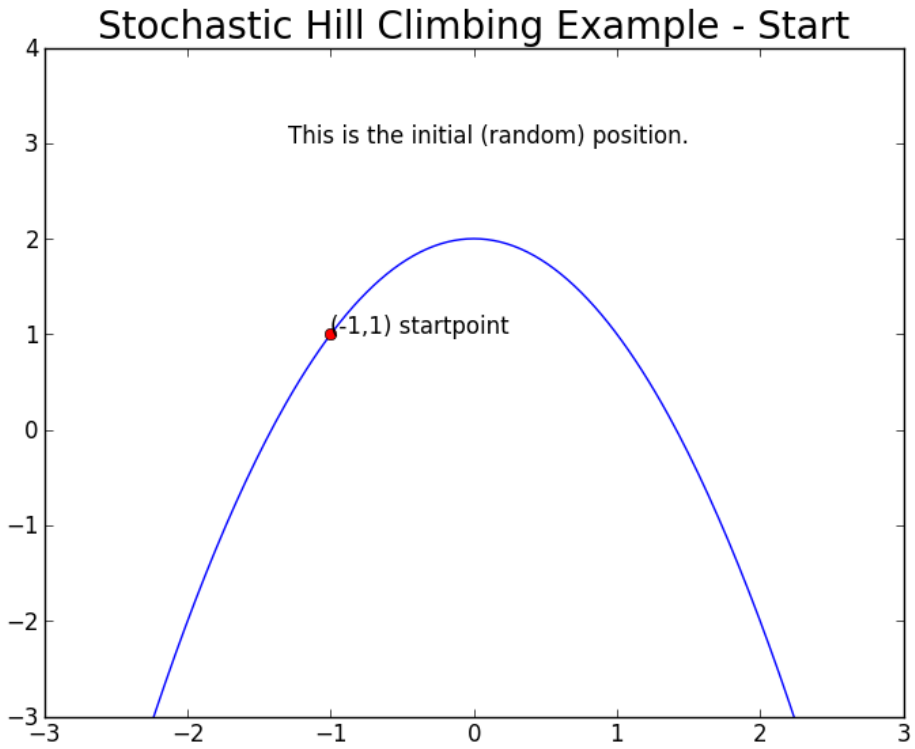

Figure 13: The startpoint for the search

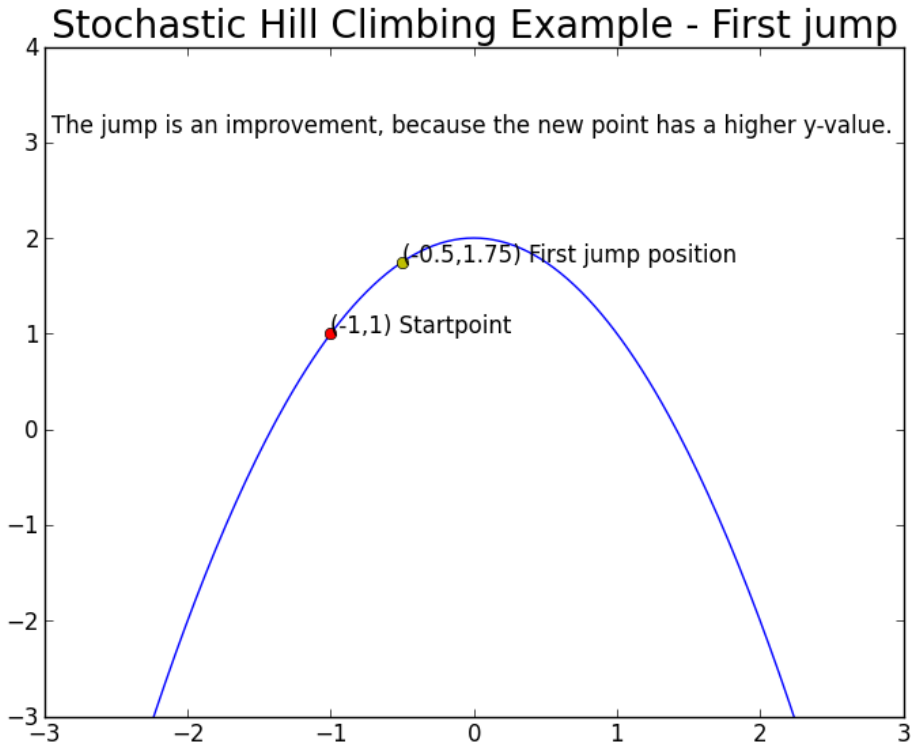

Figure 14: The first jump

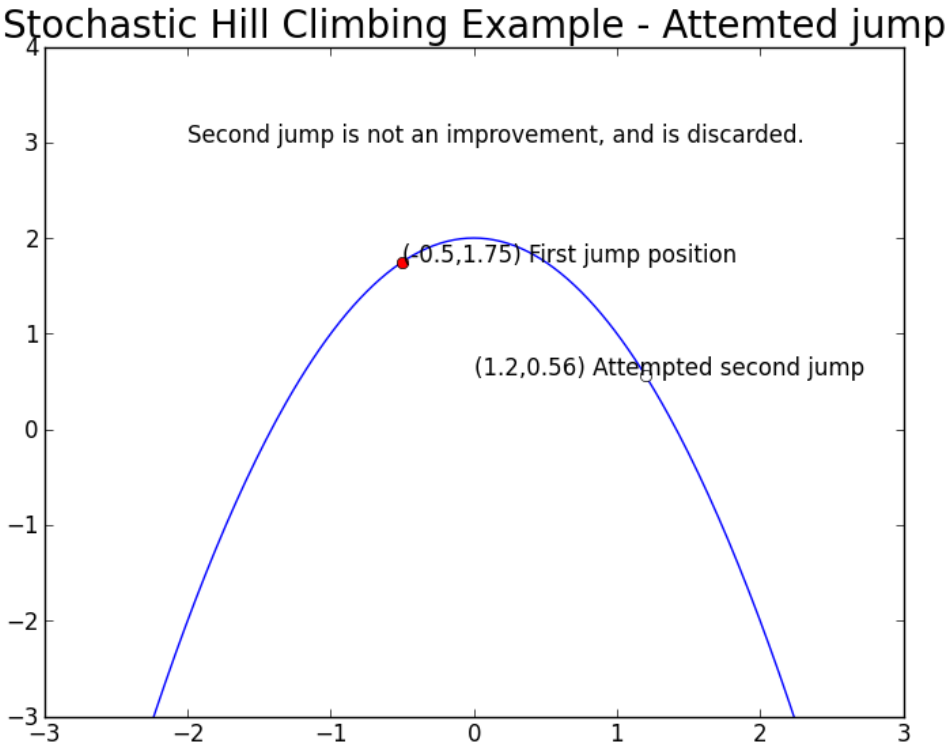

Figure 15: An attempted jump

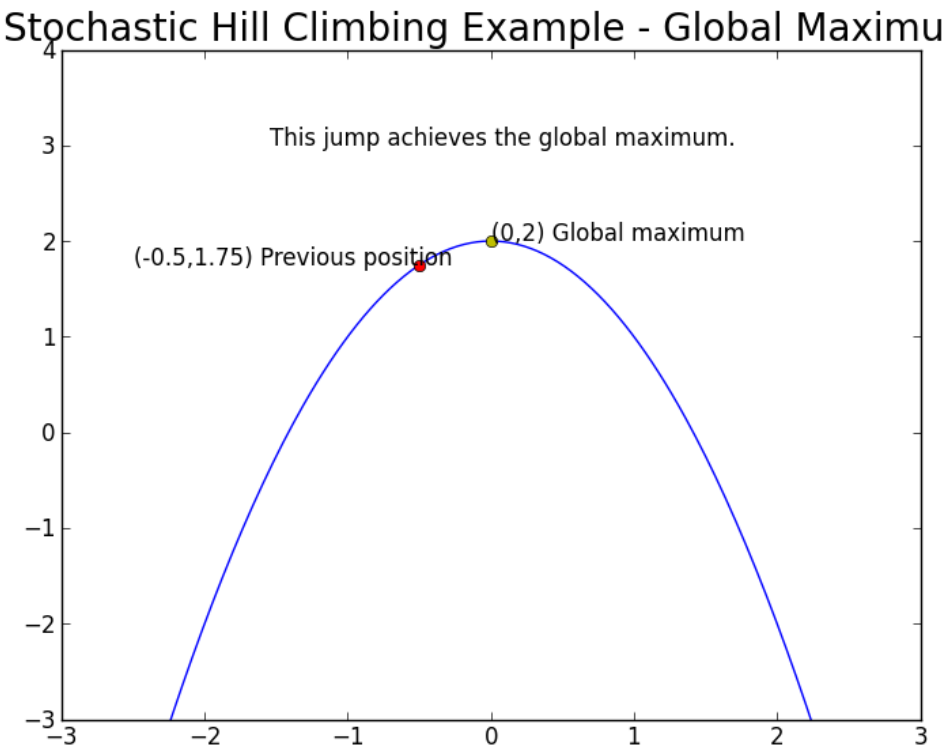

Figure 16: Achieving global maximum

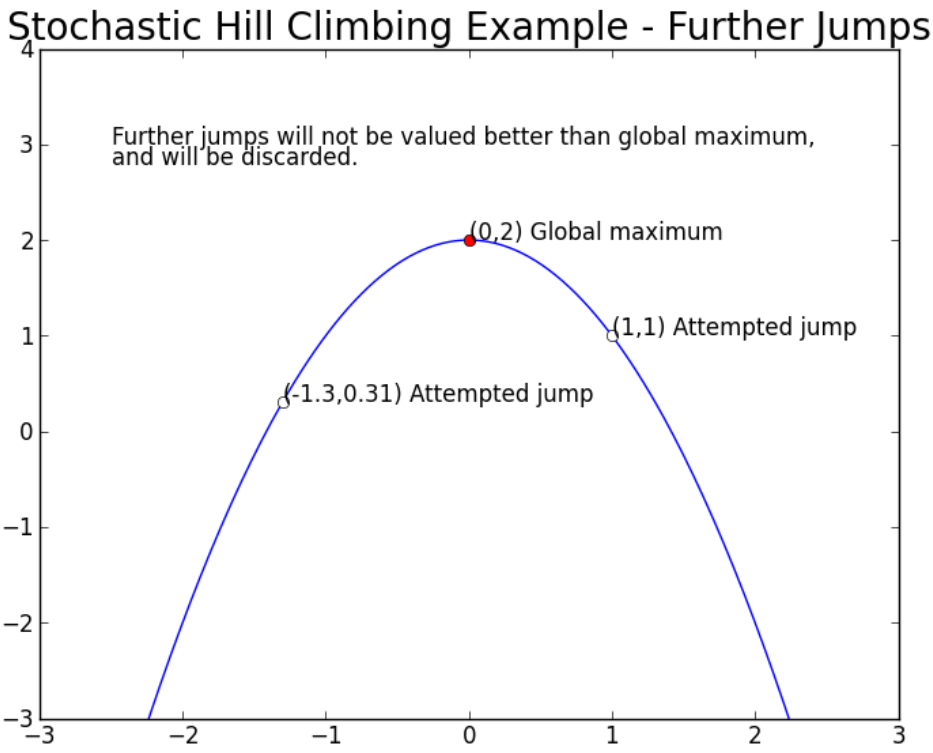

Figure 17: Further jumps are discarded

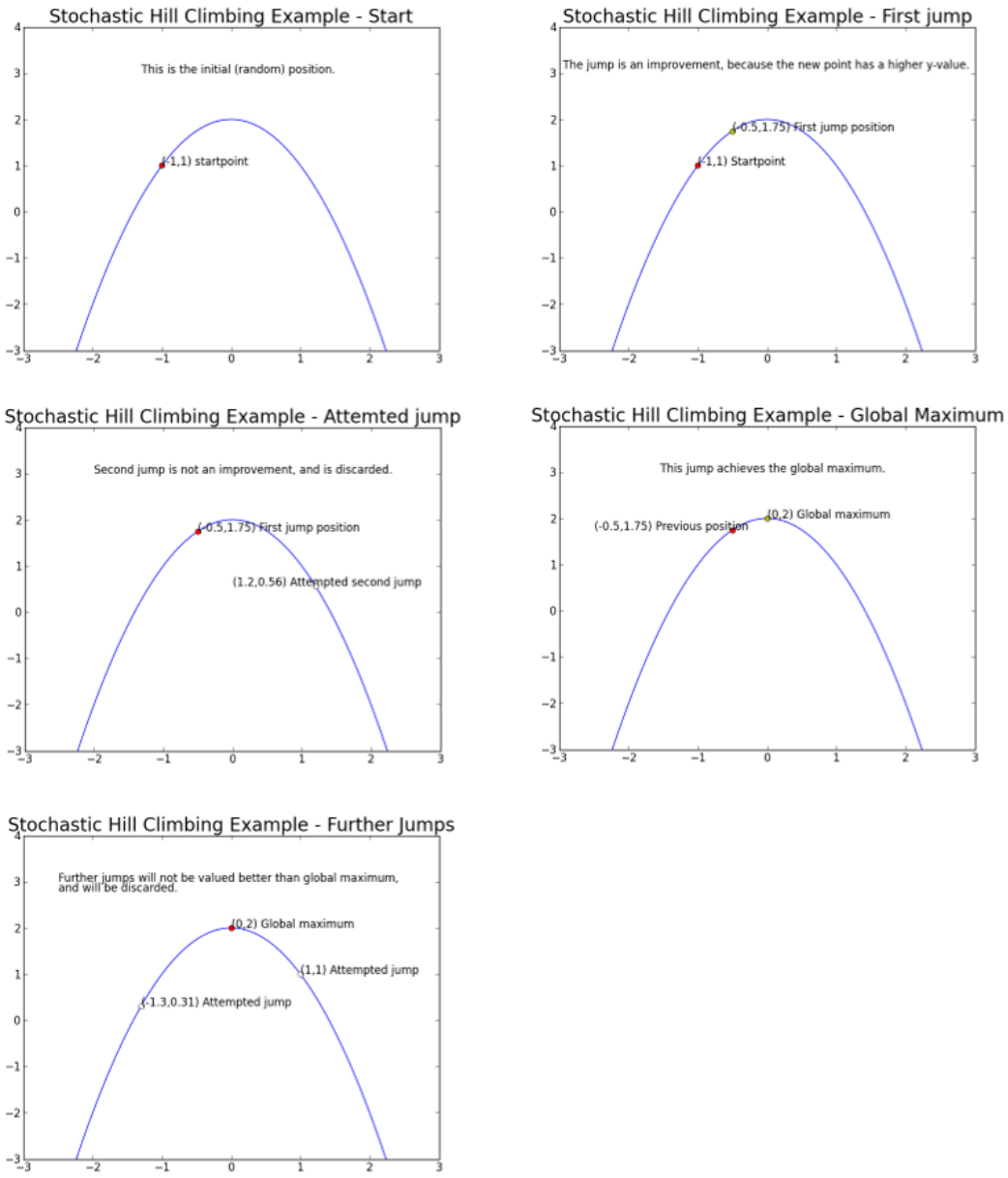

Figure 18: An example of stochastic hill climbing

This is a short example of how stochastic hill climbing will eventually find the local or global maximum on a graph. With a starting maximum jump distance of 1.0, decreasing by 0.1 with each jump, the algorithm will have converged after ten successful jumps, hopefully at an acceptable position. In an adaptive 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. In fact we found that people grow tired of being in a certain area for a long time, so we even increase the maximum jump distance when we receive the one heart feedback as described in Algorithm. This example is very short, compared to how many jumps it would normally take, but this is actually possible.

4 Solution

In this chapter we talk about our solution which is an implementation of the stochastic hill climbing 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 feeback
5       Create metric from feedback
6       if newMetric > oldMetric
7           oldBestMetric = newMetric
8           Save parameters as best known parameters
9       else
10          Restore best known parameters
11          Jump relations and parameters
12  end
```

Figure 19: Pseudocode for the algorithm.

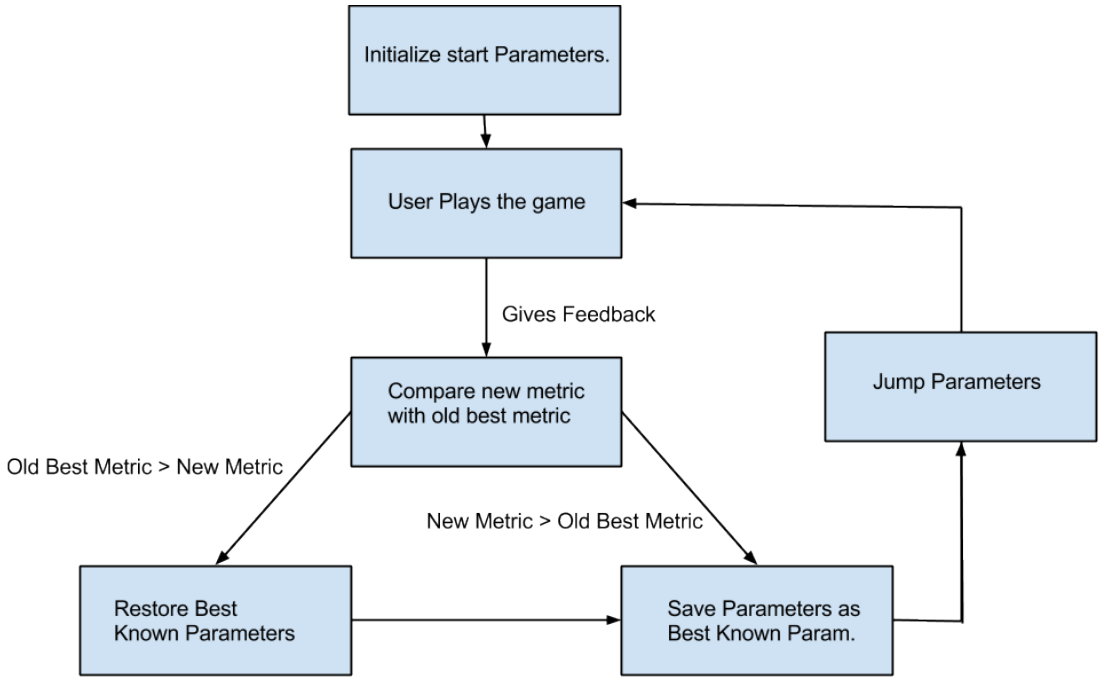

Figure 20: The flow of our algorithm.

As indicated in *Figure 19* and *Figure 20* we start with set values for all parameters, which is a prior we have found to give a decent game. After playing a round of AdaptiveTD the player gives feedback we use to create the metric. As you can see in *Figure 21* the player answers two questions after the game. The first question 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 local maximum point, as described in Stochastic hill climbing, just because the metric was assigned a high random value by the die.

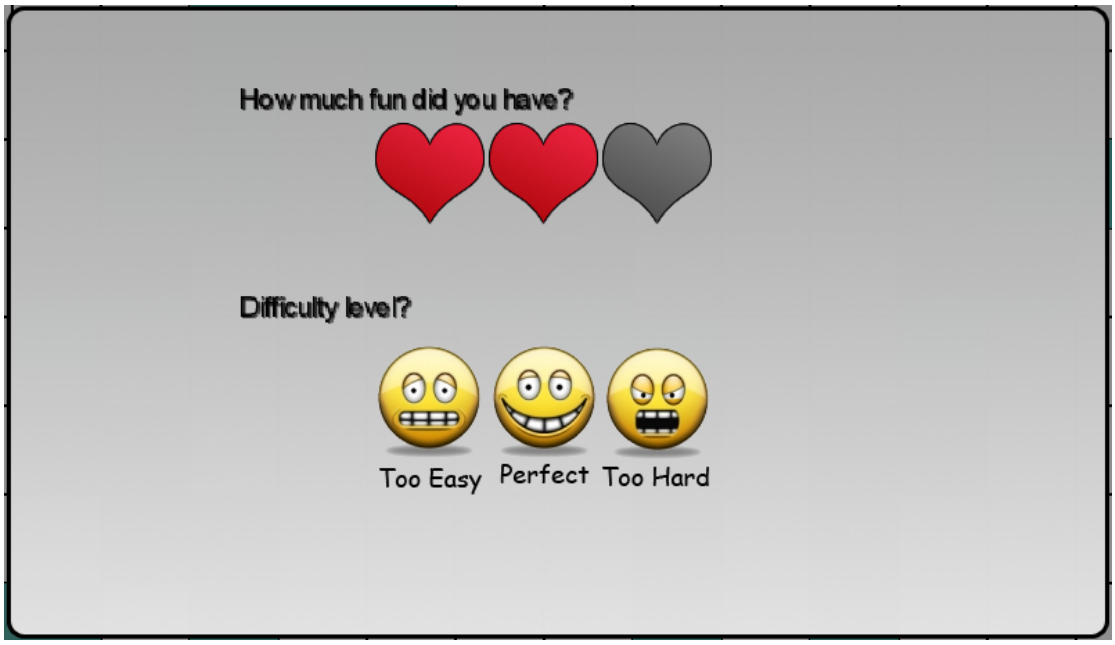

Figure 21: The questionnaire the user takes post-game.

After the metric has been created we compare it with the best known metric, and if we find that the new metric is better we save this one as the best known metric continue. Then we continue to jump from the parameters we had when receiving the new metric, else we restore the parameters from the game where we received the best known metric previously.

```
for each relation r with parameters p1, p2 ... pn
    distanceToJump = random between -maxJumpDistance and +maxJumpDistance
    find max distance we can jump without exceeding any upper or lower parameter limits
    if absolute(maxDistance) > absolute(distanceToJump) # Simplified, must check for + and -
        for parameter p in relation
            jump parameter with distanceToJump * impactFactor for parameter
    else
        for parameter p in relation
            jump parameter with maxDistance * impactFactor for parameter
end
```

Figure 22: Pseudocode for jumping of relations

We randomly increase or decrease the parameters with parameters with values from zero up to a maximum jump distance, and let the user play again and repeat the process. The maximum jump distance decreases each time we find a better game, and thus we attempt to converge at a position where the player enjoys the game. Perceived fun is fleeting though, and if the player grows tired of an area he or she might start to rate it lower. To combat stagnation we decided to increase the maximum jump distance whenever a player gives the one heart feedback as described earlier in this chapter.

As described in Parameters and Functionality, instead of just jumping all parameters randomly, as one would normally do in a stochastic hillclimbing algoritm, we jump on relations, which change the game while keeping the difficulty close to constant. We do not change all parameters individually when jumping, but cycle through all relations and jump the primary parameter in the relation, and then jump all related parameters by the distance multiplied with the impact factor.

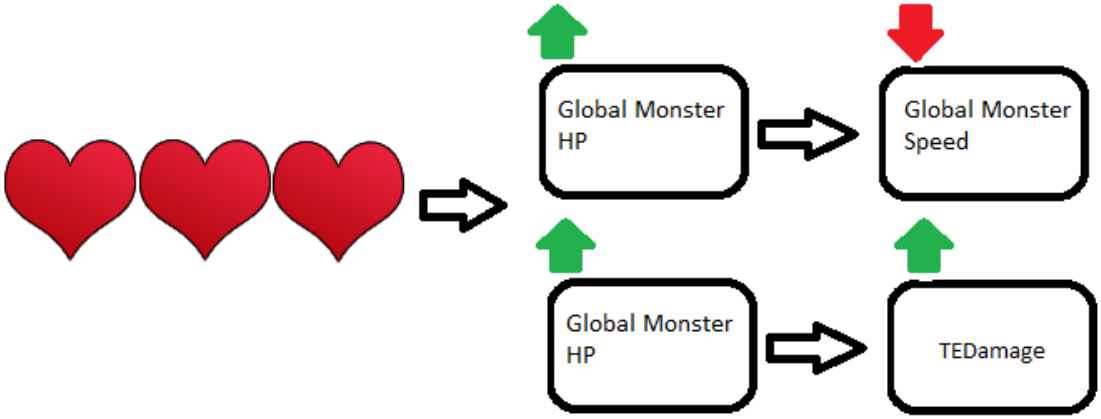

Figure 23: Basic relations example.

Figure 23 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 23* 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.

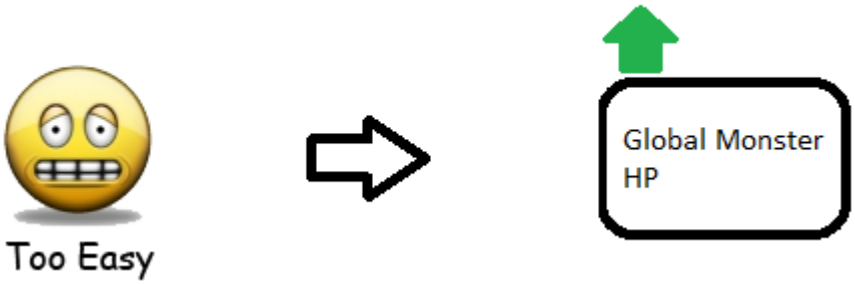

Figure 24: Difficulty does not change in relations.

We also receive feedback on the difficulty. This is not the main focus of our research, but it does indicate something about the user that we can use to adapt the game. This feedback would become increasingly relevant in future projects, if we attempt to detect the player's preferences without a questionnaire. We will talk more about this in Future Work.

Since a game that is too easy or too hard is not fun, we found that we needed to implement some sort of difficulty scaling as well. We have implemented a simple algorithm which changes a few parameters for the difficulty scaling. Since the feedback here is accurate, we can easily move the difficulty in the direction the player desires. To do this we change a few parameters, without impacting their relations such as to actually change the difficulty.

The simple algorithm that controls the difficulty is a form of hill climbing algorithm, but here we know in exactly what direction the player wants us to go due to the accuracy of the feedback.

```
distanceToMove = absolute(random(-maxJumpDistance, maxJumpDistance))
if too Easy
    health += distanceToMove
    speed += distanceToMove/2
    move one other parameter with a very low chance for each
    in the direction that makes the game harder
else if perfect
    Do nothing
else Too hard
    health -= distanceToMove
    speed -= distanceToMove/2
    move one other parameter with a very low chance for each
    in the direction that makes the game easier

check that all parameters are inside bounds still, else set to min or max
```

Figure 25: Pseudocode for the difficultyscaling.

The additional parameters we change all have a 1/6 probability of being modified, and are as follows:
- EarthquakeChance
- DiggerChance
- SuperChance
- TEDamage
- GlobalTowerRange
- GlobalMonsterGoldYield

Further description of what each does can be found in Appendix I - Parameters.

5 Testing

This chapter contains the tests we performed to see how well the algorithm worked. One of the main problems with testing is that it is hard to measure fun, so from actual players we have gathered feedback in general, rather than just asking them to rate the game. The algorithm was tested using case tests as well, which contain tests where we attempt to maximize a value by playing and giving the questionnaire proper feedback.

5.1 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 becomes 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.
- Changes make AdaptiveTD more fun, but the game has too little content to keep the player entertained for a long time. Even if the enemy health and speed is in a preferable range, it does not make you want to spend hours playing, due to the lack of progress and content.
- The difficulty scales well, and the player is able to feel the changes quickly, should they be dissatisfied.

There are also some features that certain people enjoy while others do not, and since the jumping is currently random for 15 parameters the player may receive some, but usually not all, of his or her preferences. The main thing we heard was that the diggers, as described in Parameters and Functionality can quickly become overwhelming if the player gets an unlucky jump while the maximum jump distance is high. The player should generally end up with the best known parameters if they did not enjoy the game, but there are cases where one roll of the die from one heart can give a metric higher than the best known metric, especially due to the multiplication from the GameLengthMultiplier, which increases as you play more games. One solution we found, was to never pick the current parameters as the best known if the player rated with one heart. However, we have not had the time to do real playtests using this, but believe it would give players a better experience. We tested this in the case tests below, after we received this feedback. At least numerically it gives much better results, and we hope that players testing the game in the future, will enjoy it as well.

5.2 Case tests

To test how well the algorithm worked, we test if the algorithm is able to achieve maximum or minimum values in certain parameters. We created a way to automate testing, such that when the parameter(s) in question moved in the direction we desired we rated the game three hearts, and one heart if it went in the opposite direction. For each test we ran 100 rounds of 30 games each.

5.2.1 Case test: Global Reload time

In this case test we attempted to maximize global reload time. We basically wanted the GlobalReloadTime parameter to increase to 3.0. In *Figure 26* is a plot of its value through the tests.

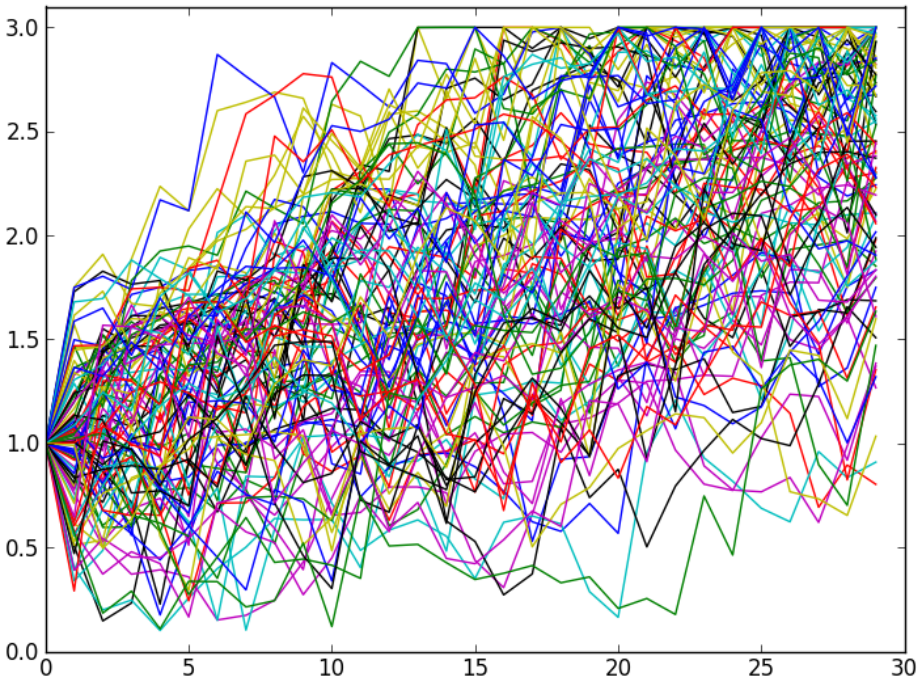

Figure 26: Plots of reload time.

As you can see, the general trend is upwards, but it seemed to be slightly to random. To fix this we thought to implement one of the things we thought about when running playtests, which was that when giving one heart ratings it is never chosen as the current best, regardless of the metric that is generated from the roll. This resulted in the following graph over 100 rounds of 30 games each.

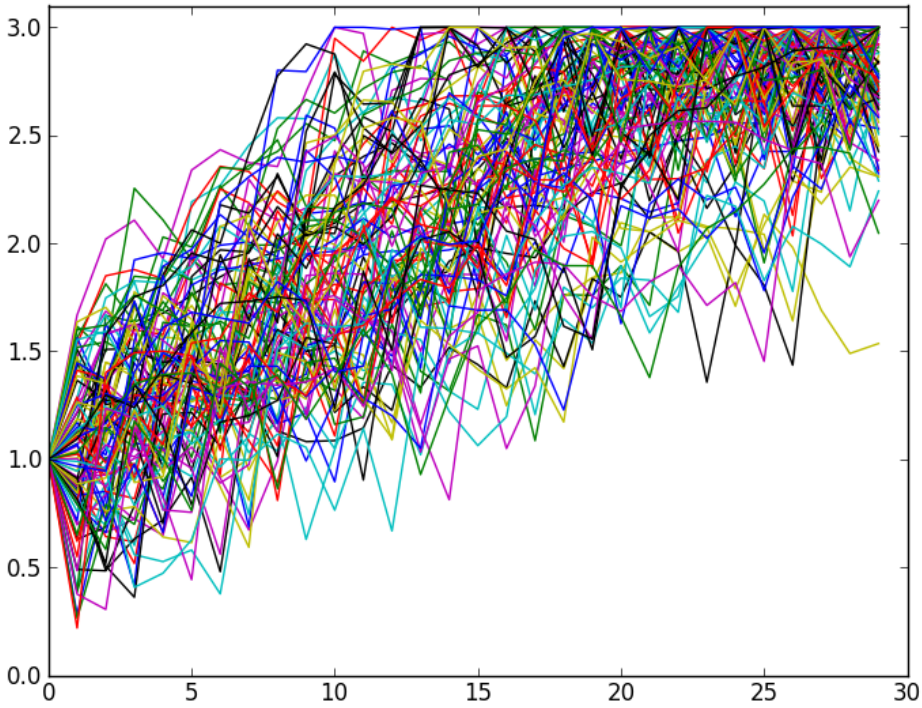

Figure 27: Plots of reload time.

The trend is that global reload time increases much more quickly. Additionally, the fact that it never picked one heart games as the current best should result in players not getting stuck with games they disliked. *Figure 28* shows the average of both through the 100 tests, here we can clearly see the difference in efficiency.

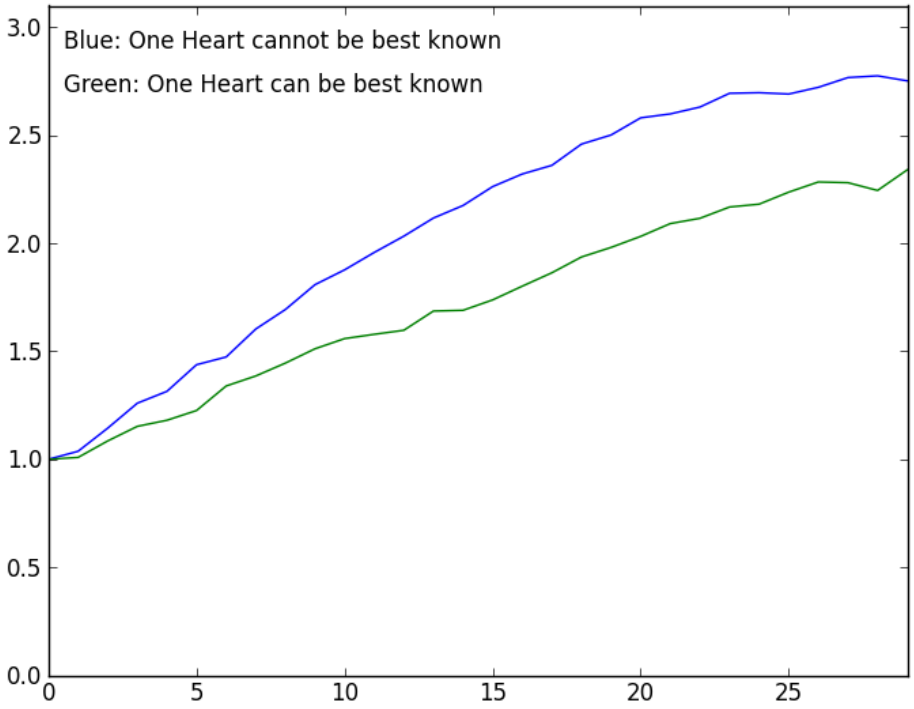

Figure 28: Plots of the average for both tests.

We also used these tests to see that the relations work correctly. GlobalReloadTime affect TEDamage, such that if GlobalReloadTime goes up, TEDamage should also increase. Here is a plot of GlobalReloadTime and TEDamage during the 100 games played after we made games with heart value one unable to be the best known game. GlobalReloadTime is the blue graph and TEDamage is the green graph. The parameters are also affected by other relations, which is why there are deviations in the values instead of always being the same.

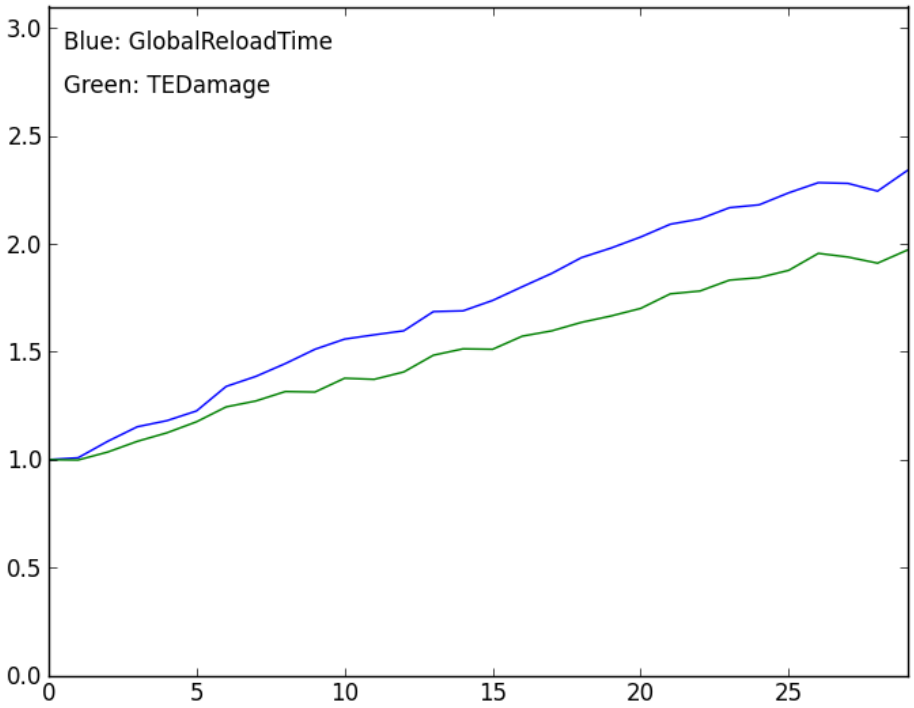

Figure 29: Plot of how global reload and tower damage changed over time.

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 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 11*, figure in the middle.

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.

A problem with the relations, is that a parameter which is represented in many relations will have a higher probability to increase or decrease more drastically, as each relation gives every parameter it holds a chance to jump. Also, our predefined relations are not completely preserving the game difficulty at each jump, making the fun search of our algorithm also slightly change the game difficulty at the same time.

The diggers, super enemies and earthquake functionality is probably what makes AdaptiveTD

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. Referere til flat:Løses ved annen mer visuell genre

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.

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.

Use search algorithm also for relations. We have most of the implementation ready for this, but the theory behind is a little more complex than the scope of this project.

9 References

[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

-Enemy information---

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