Learning Player Behavior For The Exergame BalanceFit Using Evolutionary Algorithms

Erlernen von Spielverhalten für das Exergame BalanceFit mittels evolutionärer Algorithmen
Bachelor-Thesis von Stefan Lüttgen, Matr.-Nr. 1574357
Tag der Einreichung:

1. Gutachten: Prof. Dr. Johannes Fürnkranz
2. Gutachten: Christian Wirth

TECHNISCHE UNIVERSITÄT DARMSTADT

Department of Computer Science
Knowledge Engineering Group, Serious Games Group

Learning Player Behavior For The Exergame BalanceFit Using Evolutionary Algorithms
Erlernen von Spielverhalten für das Exergame BalanceFit mittels evolutionärer Algorithmen

Vorgelegte Bachelor-Thesis von Stefan Lüttgen, Matr.-Nr. 1574357

1. Gutachten: Prof. Dr. Johannes Fürnkranz
2. Gutachten: Christian Wirth

Tag der Einreichung:

Erklärung zur Bachelor-Thesis

Hiermit versichere ich, die vorliegende Bachelor-Thesis ohne Hilfe Dritter nur mit den angegebenen Quellen und Hilfsmitteln angefertigt zu haben. Alle Stellen, die aus Quellen entnommen wurden, sind als solche kenntlich gemacht. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Darmstadt, den 31st January 2017

(Stefan Lüttgen)

Abstract

This thesis investigates the potential of learning player behavior for the game BalanceFit[1] using evolutionary optimization. BalanceFit is a scholastic exergame controlled by shifting one's center of mass while standing on the WiiFit Balance Board. In this way, the player tilts a floating playing board to eventually maneuver a ball through a labyrinth into a given finish zone. BalanceFit is used to help rehabilitating patients with impaired movement in a playful environment.

The game contains an AI (artificial intelligence), which is able to play the game autonomously. The AI's playing style is influenced by a set of parameters. Subject of this work is to investigate if these parameters can be tweaked in a way, that the AI plays according to a human example. First it is examined if the current AI design in combination with an evolutionary algorithm allows optimizing it's parameters to come closer towards playing like another instance of the AI. Afterwards it is explored if the AI is also capable of approximating human playing behavior. The motivation behind this is, that sufficiently enough approximated playing style could allow designing levels and game situations for triggering certain movements of a patient's very playing style, but without constantly needing the assistance of the patient when creating the level.

In the conducted experiments we show, that learning playing behavior of a different AI is possible in general, whereas learning human playing behavior is only successful in specific cases. Thus learning human playing behavior is possible in principle, but the current design of the AI does not allow approximating a human playing style sufficiently enough at the moment. Additionally an evaluation was made to show which parts of the AI design are most responsible when approximating a human player. Re-designing the current AI setup might achieve a more general approximations of human playing behavior.

Zusammenfassung

Diese Arbeit untersucht das Lernen menschlichen Spielverhaltens für das Spiel BalanceFit[2] mittels evolutionärer Algorithmen. BalanceFit ist ein akademisches Exergame, das durch das Verschieben des Körperschwerpunktes des Spielenden auf einem WiiFit Balance Board gesteuert wird. Dadurch wird ein schwebendes Labyrinth gekippt, um einen darauf befindlichen Ball in einen vorgegebenen Zielbereich zu manövrieren. BalanceFit wird zur spielerischen Rehabilitation von Patienten mit Bewegungseinschränkungen eingesetzt.

Das Spiel enthält bereits eine KI (künstliche Intelligenz), die selbstständig das Spiel spielen kann. Der Spielstil der KI wird durch eine Menge an Parametern beeinflusst. Thema dieser Arbeit ist es zu untersuchen, ob diese Parameter dahingehend angepasst werden können, so dass die KI gemäß einer menschlichen Vorlage spielt. Dazu wurde zunächst untersucht, ob die KI in ihrem aktuellen Aufbau mit Hilfe eines evolutionären Optimierungsverfahrens in der Lage ist, sich durch entsprechende Paramterjustierung dem Spielverhalten einer anderen KI-Instanz anzunähern. Danach wurde untersucht, ob die KI auch in der Lage ist das Verhalten eines menschlichen Spielers anzunähern. Hintergundidee ist, dass genügend angenähertes Spielverhalten eines bestimmten Patienten das Design von Spiellevels zur gezielten Stimulation bestimmter Bewegungen erlauben könnte, ohne jedoch den Patienten beim Erstellen der Level permanent zu Rate ziehen zu müssen.

In die durchgeführten Ergebnissen zeigen wir, dass das Lernen von Spielverhalten ander KI-Instanzen allgemein möglich ist. Das Lernen von Spielverhalten menschlicher Spieler ist lediglich in speiziellen Fällen möglich. Daher ist das Lernen menschlichen Spielverhaltens prinzipiell möglich, jedoch erlaubt der momentane Aufbau der KI keine allgemein zufriedenstellende Annäherung an beliebige Spieler. Zusätzlich wurde eine Auswertung angefertigt, um zu zeigen welche der verschiedenen Komponenten der KI beim Lernen von Spielverhalten den größten Einfluss haben. Eine Überarbeitung des KI-Designs könnte ein allgemein besseres Lernverhalten beim Annähern menschlicher Spieler bewirken.

Acknowledgement

Thanks to Christian Wirth and Johannes Fürnkranz for the machine learning background knowledge. Thanks to Sandro Hardy for the BalanceFit part and to Felix Biemüller, creator of the AI basis. Also to Polona Caserman and Constantin Franz, the developers of the BalanceFit game during the time this thesis was written.

[1] Developed at Multimedia Communications Lab of Darmstadt University of Technology, see Section 3.2 on page 7.

[2] Entwickelt am Multimedia Communications Lab der Technischen Universität Darmstadt, siehe auch Abschnitt 3.2 auf Seite 7.

Contents

1 Introduction

The introduction presents the motivation for this work, explains the structure of this thesis and formulates the goal, the object of labor.

1.1 Motivation

Serious games, and especially health games, are becoming important aids when it comes to either preventing or rehabilitating human individuals. An exergame is specifically designed to make the player physically exercise when it comes to playing the game. These games can be used in prevention and rehabilitation of movement disorders by triggering and tracking specific body areas of a patient. However, the game has to be adjusted to the patients needs in order to have a soothing effect. If the patient is either unchallenged or overchallenged by the game, the effect is lessened.

One important part in achieving this goal is an appropriate designed game with respect to the patients needs. In order to do this, working together with the patient directly is the intuitive, but often infeasible approach, because patients would most of the time be serving the purpose of adjusting the game, rather than playing it. A method to create a level without the patient being present, is always of huge interest to the game designer. Having the artificial intelligence evaluate the labyrinth first, would allow designing a customized labyrinth without the patient's aid.

The idea is to let the player play a set of levels, which can then be used to derive their certain style or way of playing. Once a players style is learned, levels can be designed without them being present. Thereby the therapist is able to generate a level, which could satisfy the patients needs for triggering special movements within their treatment.

1.2 Structure of this Thesis

This thesis can be divided in three parts: a theoretical part from Chapter 2 to 5, an application or experimental part in Chapter 6 and a results respectively conclusion part in Chapter 7. Chapter 8 summarizes this work and gives an outlook regarding future endeavors.

The beginning deals with related work, an introduction to the game BalanceFit and describes the existing control based artificial intelligence system. Then the related fields of machine learning, optimization and evolutionary algorithms follow. The application part thereafter describes the experimental setup and the results from these. After an evaluation part and a summary, this work closes with an outlook regarding future endeavors and potential continuations of this work.

1.3 Goal

The goal of this work is to learn a human playing style for the game BalanceFit. This allows predicting possible behavior when it comes to certain situations in the game. Therefore it is investigated if the already existing control based artificial intelligence system in the game is able to learn a playing style of a particular player. Employing a genetic optimization method, we first want to learn the playing style of a different instance of the AI itself, and afterwards try to learn by the example of a human player. Depending on the quality of the approximated playing behavior, this would allow designing levels and game elements to trigger certain movements related to a patient's health status.

2 Related Work: Learning from Demonstration

This chapter examines the work already done in the field of having an artificial intelligence learn behavior from demonstration, especially in the field of games. The learning process can be divided into two sections. The first one, where the computer learns by playing only against itself, and the second one, where the computer is playing against an expert. The expert can either be a skilled human player (respectively data drawn from a human player), or another computer, which can already apply a winning game strategy.

A common feature is the usage of *reinforcement learning*. This optimization approach is suitable when it comes to an agent interacting with its environment and especially improving from these interactions [41]. For the representation of the optimization problem often times a *Markov decision process* is used, which is also used as an underlying representation in this work[3].

Using self-play as a learning environment has the advantage, that it is not relying on outside help or databases. On the other hand it might create a slowed down learning process by playing unusual or random situations [18].

For example, reaching the strength of an expert human player in a board game was achieved using reinforcement learning to learn the board game backgammon in 1995. Apart from the playing ability, the program even found a particular opening strategy, making contribution to backgammon theory [42]. Here it comes in handy that a self-playing algorithm doesn't refrain from unusual positions, allowing the algorithm to examine unusual or unknown game situations.

More recent results have shown that the a computer is able to learn perfect play in simple computer games beyond human strength by only having raw screen data provided, and employing self-play shown by researches of Google DeepMind in 2013 [28]. Here, the computer was not only able to learn perfect play, but also found winning strategies which were not known to the designers thus far.

In contrast to self-play, learning from demonstration means learning from players of some skill respectively from players who have a better game strategy than random play. This can either be a human or computer program, but also a database of games played by experts.

A similar experiment as the previous mentioned example of backgammon was conducted with the game of chess. Also using reinforcement learning, here the program KnightCap was playing against humans over the Internet and achieved human expert strength within three days of training in 308 games [2].

A project starting in 2015 by Google DeepMind named AlphaGo is dedicated to developing an artificial intelligence to play the game of Go [38]. Go is a Chinese board game and known for its higher complexity than chess. Therefore it was only in 2016 that DeepMind's AlphaGo was able to beat a human professional master player under official tournament circumstances for the first time. Alpha Go was trained using databases, which can be seen as learning from demonstration. Afterwards it continued training by playing against other instances of itself.

In our experiments we also first let the AI learn from other instances of itself before we move to learning from human examples. Similar to the above mentioned work, we are also relying on a representation based on Markov decision processes.

In contrast to the aforementioned work, we use evolutionary optimization as an optimization method in our learning process. This is because we cannot say if a solution, in case we find it, is a unique solution, or if there exits a better one, which we just did not find yet. Therefore we assume a non-convex optimization problem. Furthermore, since the BalanceFit AI was not subject to any investigation yet, we want to make as least assumptions about our optimization problem as possible prior to the optimization process. Evolutionary optimization is qualified in finding solutions in these kinds of optimization problems[4].

[3] For MDPs see Section 5.3.1 on page 16.

[4] On exploring a non-convex search space see Section 5.5.4 on page 23.

3 Serious Games, Exergames and the Balance Fit Game

This chapter gives an introduction to the field of serious games and describes the exergame BalanceFit in detail. Afterwards the controlling of the game is explained. Then the use and goals of BalanceFit as an exergame with respect to serious games in general is presented.

3.1 Serious Games and Exergames

Serious games are games designed to serve a different purpose other than only entertainment, which is indicated by the *serious* part in its name. Serious games connect the purpose of "learning goals of a given educational objective"[34] to video game technology. These objectives include the fields of health care, scientific exploration, politics, planning or education in general.

As early as 1812 a Prussian simulation, called *war game*[5], was developed to train strategic skills and tactical maneuvers. The first video game adaption of this idea arose during the cold war [14]. After *Tennis for Two*, considered as one of the first video games and originally displayed on an oscilloscope, parallel to pure entertainment games, soon educational games were published. For example *The Oregon Trail* in 1971, a history teaching game with focus on the American pilgrimage in the 19th century [44], or *Captain Novolin* in 1992, a health game teaching the way of dealing with diabetes [20].

In the professional sector, as military or engineering, simulations are indispensable. A commercial pilot is usually training in a flight simulator before controlling a plane. The military uses serious games for assessment as well as training purposes. Companies in an economy can utilize serious games for recruiting or assessment in general, because of their ability to produce quantitative and comparable results [26].

In the civilian sector serious games are a fruitful aid serving an educational purpose by requiring active participation and encouraging interest and engagement[21]. Especially in rehabilitation, games which require the player to physically move, so called *exergames*, allow the game designer to track and analyze movements, when the player controls the game.

3.1.1 Exergames

As a subtype of serious games, exergames[6] make use of body movement and reaction tracking devices. One of the most famous examples, although most famous in Japan, is the game *Dance Dance Revolution*, introduced by Konami in 1998, where the player has to touch certain areas on a carpet-like game pad, which implicitly makes them resemble dancing moves. In Europe exergames became famous with the introduction of the Nintendo Wii game console in 2006. The games there in the initial version are controlled using a remote, which has an accelerometer to track movements. In this way, games like tennis, golf or bowling can be played by imitating the actual movement of the particular sport.

Other devices are the Wii Balance Board, further explained in 3.3.1, or visual tracking devices such as the Microsoft Kinect camera[7]. However all these devices have in common that the player has to move their body in order to send an input to the game. Therefore exergames are particularly suitable in coping with people suffering from physical symptoms by either testing their abilities or forcing them to make certain movements in a playful environment. Children and younger people are naturally pleased by this form of therapy, but also older people benefit from this more comfortable user experience [30].

Apart from the therapeutic advantages, exergames also refute to some extent the reputation of video games as an inactive engagement as part of the sedentary lifestyle in first world countries [39].

3.2 Elements of BalanceFit and Goal of the Game

In BalanceFit the player is faced with a labyrinth on a floating game board, which can be tilted around it's center. This sets a ball, located on the surface of the game board, in motion, following basic mechanical laws of the inclined plane. The goal is to tilt the board in a particular fashion, such that the ball is maneuvered through the labyrinth and eventually into a certain area of the board, namely the goal.

[5] The original German term is *Kriegsspiel*.

[6] Combination of *exercise* and *game*.

[7] Originally introduced for the Xbox 360 gaming console, it is nowadays also widely used as a 3D full-body motion tracking device.

3.2.1 Level Design and Board

The different levels in the game are automatically created to challenge the player according to their skills using *procedural content generation*[19] [29]. The main element is the board itself, but the design theme is also adjustable in order to make it appealing to the player. An example of the board, start and goal and the ball in a basic labyrinth in a forest-like theme, called *bee world*[8], can be seen in Figure 1.

Figure 1: The board and the ball in a basic labyrinth. The flowers in the upper right corner and in the center indicate the start and the goal zone respectively.

Further level elements are objects, which let the player lose directly when touched, for example holes in the board (in the *wood world*) or ball eating plants (in the *bee world*).

3.2.2 Goal of the Game

The goal of the game is to maneuver the ball into a goal zone. There are several ways to not achieve this goal a.k.a. losing the game:

- Tilting the board in a way which makes the ball leave the playing area and let it dive into the endless abyss, where the board is floating in.
- Directing the ball into a hole or a ball eating creature.
- Losing on time. There is a clock running in the background which makes the player lose if they exceed a set time limit.

3.2.3 The Ball and the Physics Engine

The movement of the ball is following the Unity physics engine, which the game is based on. In the 3-dimensional space the middle of the board is located in the origin of the world coordinate system. The board angle, ball position and ball velocity are indicated by 3-dimensional vectors $(x, y, z)^\intercal \in \mathbb{R}^3$, where x indicates the intercept in north to south direction, y indicates the height and z the intercept in west to east direction, when facing the board during the game:

- **Board angle** $(x, y, z)^\intercal_{angle}$:
  The tilt of the board is described by the deviance from the origin in the x, y and z direction.
- **Ball position** $(x, y, z)^\intercal_{pos}$:
  The position of the ball is simply described as a coordinate triple $x, y, z^\intercal$, where y usually is close to 0 as long as the ball is located on the board.

[8] The other available theme by the time this thesis was written is the so called *wood world*.

- **Ball velocity** $(x, y, z)^{\intercal}_{vel}$:
  The velocity of the ball is described by the different velocities in the x, y and z direction and computed by the dot product of its $(x, y, z)^{\intercal}$ vector.

Among other physical properties of the game elements, these three are in particular of interest to us. While for controlling the game only the board angle is necessary, because the ball position and its velocity are computed implicitly by the physics engine, for the optimization process, additionally to the board angle, the ball's position and velocity are also taken into account[9].

3.3 Controlling the Game

The player's endeavor is to maneuver the ball through a labyrinth to a position within the goal area. This is done by altering the board's angle to induce a movement of the ball, either by using a keyboard, the accelerometer of a tablet or smartphone, or while standing on the Wii Balance Board.

3.3.1 The Wii Balance Board

The Wii Balance Board was originally developed by Nintendo for the Wii Fit game series for the Nintendo Wii game console as an accessory controller and introduced in 2007. The control of the game is done by moving one's body in order to shift the center of mass perceived by the weight sensors of the board, similar to a force platform. The board has four pressure sensors detecting weight alterations in the four corners of the board by measuring the force for every pressure sensor in newton. Thereby the player's center of balance can be compute by detecting the player's center of mass over the board's surface [33].

In Figure 2 the process of the alterations of the center of mass while playing a level in BalanceFit is shown for two different players.

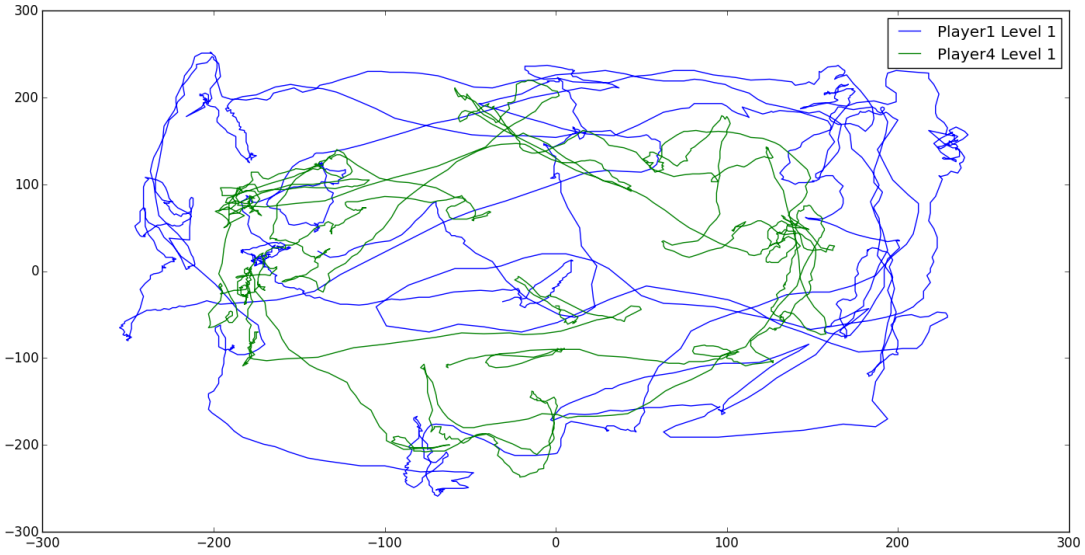

Figure 2: The shift of the center of mass over the period of playing level one in the wood world. Note the difference between the two players when alternating their center of mass.

The board provides the change of the center of balance with regard to the four corners of the board. The change of the board angle in x and y direction is then determined by the formula

$$x = tr + br - (tl + bl)$$

[9] See Section 5.3 on page 16.

for the x-coordinate and

$$z = tr + tl - (br + bl)$$

for the y-coordinate of the board, with tr depicting the pressure detected by the top right sensor, br the bottom right, tl the top left and tr the top right sensor of the Wii Balance Board. The height y of the center of the board is constant, such that the board is tilted in $x$and z direction around its center, which is located on the origin of the game's coordinate system.

Only playing the game in connection with the Wii Balance Board lets the player experience the full spectrum of the exergame aspect of BalanceFit.

Figure 3: A person playing BalanceFit using the Wii Balance Board. The change of the body's center of mass to the left (from the player's point of view) can be seen as as a sign, that the game in connection with the Wii Balance Board serves the purpose of physically engaging the player.

Similar to the Microsoft Kinect 3D-Camera, the Wii Balance Board has also become a measurement tool widely used in scientific research [11], although originally developed as a game controller.

3.4 BalanceFit in the Context of Serious Games

BalanceFit was developed by the Serious Games Group at the Multimedia Communications Lab at the Department of Electrical Engineering and Information Technology of Darmstadt University of Technology. It was created to encourage patients to activate certain muscle areas in an attractive and playful environment. Since 2010 it is used to record data of patients suffering from imbalances in muscle activation due to nervous disturbances as for example after apoplexy or multiple sclerosis.

Another field of application is the rehabilitation of bodily impaired juveniles. Here the aspect of playing a video game happens to be particularly appealing to younger people nowadays.

4 BalanceFit PID Controlled Artificial Intelligence

This chapter describes the inner workings of the artificial intelligence in the game BalanceFit, which was developed by Felix Biemüller during his Bachelor thesis at Darmstadt University of Technology in 2015 and is described in detail in [6]. First it is shown how the AI plans its way through the labyrinth. Afterwards it is focused on how the board tilt is achieved in order to maneuver the ball along the planned path. Ultimately an overview is given on the different parameters the artificial intelligence uses.

4.1 Pathfinding

The pathfinding algorithm, described in [13], is based on Dijkstra's algorithm [12] for finding the shortest path in a graph. The level is represented as a matrix Z^{mxn}, with entries $z \in Z_{floor}, Z_{wall}, Z_{hole}, Z_{start}, Z_{finish}$, describing the setup of the current level with respect to every cell. These cells are weighted by a cost factor w, such that the algorithms can employ strategies to avoid holes or prefer walls. These weight factors decrease with increasing distance. For example a hole's weight w_{hole}^{d-1} has the actual weight factor assigned to the adjacent cells, and is exponentially deceasing, the higher the distance d to the cell is, for $d \in \mathbb{N}$ indicating the distance in number of cells between the actual cell and the hole itself. The resulting cost factor of diagonal cells are multiplied by $\frac{1}{\sqrt{2}}$.

In [6] the pathfinding was extended to take *outer diagonals*[10] as adjacent cells into account. This extends the set of adjacent cells now to a maximum amount of 16 cells[11]. Normal use of diagonals allows a change in direction of 45°, where the outer diagonals allow now an additional change of 25° resp. 65° in change of direction per cell.

Another extension is the implementation of the idea of the Bresenham's line algorithm [10], which generally finds a straight line between two points in a raster matrix, and here is smoothing the path finding way, since it can be detected if a straight line in the level matrix is free of obstacles.

Maneuvering around a hole can be seen in Figure 4, where long diagonals and long straight lines are already incorporated. The assigned weights for each cell are depicted as a heat map, where yellow refers to a square with low avoidance and red with high avoidance. Blue cells are avoided completely, such as walls, because it is impossible to cross them.

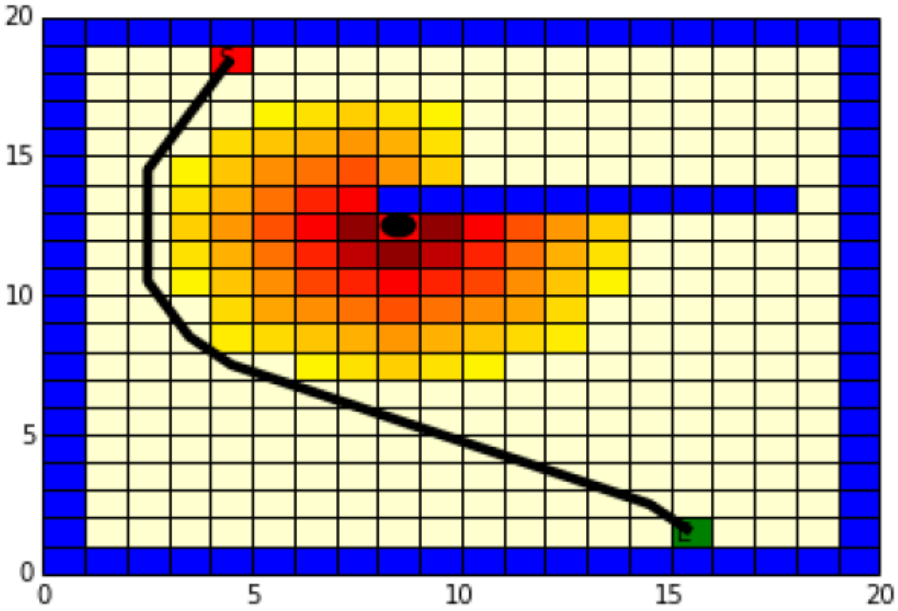

Figure 4: Finding a path from start (red) to finish (green). The black hole is avoided due to the weights assigned to its neighborhood, indicated by the heat map.

[10] Similar to Knight moves in chess.

[11] Objects or walls cannot be jumped over, therefore often less than 16 cells are taken into account as the adjacent cells including long diagonals.

4.2 PID Controller

After the AI found a proper path and the cells to visit are set, a strategy to alter the board's angle in order to visit these cells has to be employed. This is done using the idea of a PID controller, which continuously calculates the difference between the desired ball position and the actual ball position as an error. Minimizing this error over time by adjusting control variables, will result in a board angle controlling, which maneuvers the ball from cell to cell, following the desired path relative to a certain error.

4.2.1 PID Control Theory in Principle

A PID controller consists of three terms, the *P-term* (proportional), *I-term* (integral) and *D-term* (derivative) [1]. Usually one starts with the P-term $u_p(t)$, which basically is the error $e(t)$ of the desired setpoint[12] $r(t)$ and the actual process value $y(t)$ at time step t multiplied by the *proportional gain* parameter K_p, such that:

$$u_p(t) = K_p\, e(t) = K_p\,(r(t) - y(t))$$

The parameter K_p controls how large the output is with respect to the error. Larger output K_p means a higher response of the controller, where a smaller K_p will result in a less responsive controller. In the case of the BalanceFit AI, a large K_p would result in a big change in the board angle, a small K_p in a small change. Important here is that $u_p(t)$ is converging towards 0, such that it is possible to overshoot for large K_p, because the controller doesn't directly change the velocity, but the acceleration of the board tilt. This would result in a constant alternating movement around the desired position.

Therefore the I-term is introduced in order to take the error over time into account, or more precisely the sum of the instantaneous error over time, denoted by

$$u_i = K_i \int_0^t e(t)\,dt$$

with t as the variable of integration and K_i as the *integral gain*, which similarly to K_p just multiplies the output value. It can also be seen as the error, which should have been corrected before. While u_i responds to errors from past time steps, it can still cause the actual value to overshoot, especially when the set point $r(t)$ has a limit like a board angle of 90° in the case of BalanceFit. If $u(t)$ surpasses this value, the error is not decreasing as fast as the actual value $u(t)$ is increasing, causing the integral to increase also faster than in the case of having the desired value, the optimal case. So to have an estimation of what the error will be in future time steps, the D-term is introduced as

$$u_d = K_d \frac{de(t)}{dt}$$

with K_d as the *derivative gain*, multiplying the derivative of the error. Using the example of bringing the board into a desired angle, the idea of the D-term is obvious: the derivative of the position is the velocity, means that the error decreases, the faster the board is moving towards this angle. A negative sign creates a counter force with respect to the direction. The same works when diverging from the desired angle, resulting the control system to correct the board towards the desired position. The D-term can therefore be seen as a corrective term, which alone wouldn't bring the board into the desired position[13], but corrects the P- and I-term.

Putting the P-, I-, and D-term together, the following formula can be derived:

$$u(t) = u_p + u_i + u_d = K_p e(t) + K_i \int_0^t e(t)dt + K_d \frac{de(t)}{dt}$$

which forms the base of the board control.

[12] In control theory, the setpoint is the desired value or target value for a process.

[13] Similar to the idea of a stationary point, where a function neither increases nor decreases and its derivative equals zero.

4.2.2 Cascading PID Controller: Tilt and Balance Control

Cascade control can be used to avoid propagating an error to the other parts of the control system by correcting disturbance before it will influence the output signal. The control system is BalanceFit therefore is designed as a controller for correcting the board angle, which gives its signal to the second controller, which is responsible for simulating the balance control. The next section will give a detailed view on the relevant parameters.

4.2.3 Parameter Overview

This an overview over the parameters the AI is using to play the game as well as a short description of how they affect the playing behavior. Only the parameters, which might be of interest in an optimization process are mentioned. A full overview is given in [6].

The parameters can be divided into *tilt control*, *balance control* and *pathfinding*. The tilt control variables contain the PID weights for controlling the angle of the board. The balance control variables contain the PID weights for the simulation of the player's center of mass. The pathfinding variables contain the settings for computing the path from start to end. All parameters are float values, unless specified specifically.

- **Tilt control[14]:**

K_{P-NC}	P-gain, angle of tilt.
K_{I-NC}	I-gain, compensation of ball friction and inertia.
K_{D-NC}	D-gain, precision in reaching a certain point on the board.
$deadtime$	Time between updating for desired positions, *reaction time*.
$useWalls$	Enables the usage of walls. ($\rightarrow boolean$)
$leaningAngle$	Factor the board angle is multiplied with, while leaning against a wall, resp. how much will be leaned against a wall.

- **Balance control:**

K_{P-BC}	P-gain, Intensity of weight shift.
K_{I-BC}	I-gain, Prediction of desired position in next step
K_{D-BC}	D-gain, precision in weight shift.
$maxAccel_x$	Maximum acceleration of center of mass in direction of x.
$maxAccel_y$	Maximum acceleration of center of mass in direction of y.
$maxPosX$	Maximum reachable position of the center of mass in x direction.
$maxPosY$	Maximum reachable position of the center of mass in y direction respectively. $maxPosX$ and $maxPosY$ simulate movement limitations.

- **Pathfinding:**

[14] The subscript NC refers to the German notation from [6] ans stands for *Neigungscontroller*, the German equivalent to *tilt control*.

$useDiagonals$	Takes diagonal directions of 45° for the path into account. ($\rightarrow boolean$)
$useOuterDiagonals$	Takes the outer diagonal directions of 25° and 65° for the path into account. ($\rightarrow boolean$)
$holeRadius$	The minimum distance a path will be set relative to the hole.
$holeValue$	The weight resp. cost of cells with holes.
$decreasingFactorHole$	Allows to customize the decrease of hole weighting.
$wallfactor$	Adjusts the weight of wall cells Z_{wall}, such that the pathfinding algorithm prefers or disfavors these cells.

5 Representation, Optimization and Evolutionary Algorithms, Evaluation

This chapter gives a an overview over artificial intelligence, machine learning and optimization in general and explains the terms and mathematical notation used in this context. An elaboration on evolutionary algorithms and a description of the particular method used in this work will be given, as well as the corresponding representation of the BalanceFit game and how a learning process can be evaluated.

A machine learning or optimization process is often divided in three parts: *representation*, *optimization* and *evaluation* [16]. After a general introduction on having machines algorithmically improving in accomplishing human tasks, the representation of the game as a Markov Decision Process is covered in 5.3. Afterwards the key parts of the optimization process with respect to machine learning are described in Section 5.4.2. Evaluation, meaning assessing the quality of our model, will be covered in Section 5.6.

5.1 Introduction

Making machines act or think like humans already appear in Greek mythology and throughout history of philosophy [35]. First tangible attempts were made by Ada Lovelace[15] when thinking about possible applications for the *analytical engine*[16], a machine which was in theory able to do logical computations and therefore also logical reasoning she concluded [23].

Alan Turing was one of the pioneers in artificial intelligence. Developing a chess playing algorithm himself and bringing some ideas on how to solve games using computers [43]. He also established the *Turing test*, which states that after passing it, an artificial intelligent agent cannot be distinguished from a human anymore. This created hope during the 1950's and 1960's, that thinking machines will be a reality soon. Yet major advances were only made from the late 1990's on, experiencing a second pinnacle after 2005, where neural networks seem to be able to surpass the human individual in most specific data-driven tasks [27].

Nowadays systems based on artificial intelligence are ubiquitous. May it be a navigation system in a modern car, Internet searches or auto-correction in smartphones. In many special domains these systems quickly overcome human abilities. The creation of a general intelligent system, also called *strong AI*, has not been established yet.

5.2 Terms and Notation

Terms in connection with machine learning, optimization and evolutionary algorithms:

- **Estimate**: Approximated solution according to a desired optimal solution.
- **Model**: Function producing an estimate.
- **Data Set**: Underlying set of elements to train a model in order to make predictions.
- **Sample**: Element of the data set.
- **Training set**: Subset of the data set used during the optimization process.
- **Test set**: Subset of the data set used to evaluate the model.
- **Individual**: Candidate solution.
- **Chromosome**: Properties of individuals.
- **Population**: Set of all individuals.
- **Mutation**: Altering chromosomes of parent individuals to form potential offspring individuals for the next generation.
- **Parent**: Individual generation n to form new individuals in generation $n+1$.
- **Offspring**: Mutated individuals of the generation in the next iteration.

[15] Considered as the designer of the first computer program in history after developing an algorithm for the not yet built analytical engine.
[16] Described as a general-purpose computer by Charles Babbage in 1837. A fully working version was never built.

- **Descendant**: One individual of the Offspring.
- **Fitness function**: Returns the value of a given individual. Also called error function or objective function.
- **Error**: Difference between current fitness and optimal fitness.
- **Generation**: One optimization iteration, which results in a different population. This process is looped until the resulting population meets the criteria for convergence, e. g. an error which is lying below a given threshold.
- **Selection intensity**: Controls the exploration vs. exploitation trade-off.

5.3 Markov Decision Processes and Representation of the Game

Representation, or decoding relevant data of the process one wants to optimize, such that the computer or algorithm can process it, is covered in this section. The game is represented through *state-action* pairs and maneuvering can be seen as a Markov decision process.

5.3.1 Markov Decision Processes (MDPs)

Markov decision processes allow modeling decision making processes under uncertainty, especially stochastic optimal control processes in discrete time steps [22]. Stochastic refers to an unpredictable environment, due to uncertainty or complexity, and discrete time steps refer to explicitly separable points in time[17]. Optimal control is covered in Section 4.2.1 on page 12.

Mathematically MDPs are defined as a tuple $(S, A, P_a(s,s'), R_a(s,s'), \gamma)$, with $s, s' \in S$ and $a \in A$ [3]:

- **S**: set of states, e.g. a state is a ball's position in the labyrinth.
- **A**: set of actions, e.g. an action would be tilting the board.
- **P(s, a, s')**: transition function, mapping a state-action-pair (s,a) at time step t to a probability resulting in state s'. Written as conditional probability[18] would be $Pr(s_{t+1} = s' | s_t = s, a_t = a)$, meaning that given a state s and action a, the probability of leading to state s' is returned. An example in BalanceFit would be the likelihood of a ball resulting in position s', given initial position s and a board tilt action a.
- **R(s, a, s')**: reward function, which returns the average reward when transitioning from state s to state s' using action a. In our case this can be see as the similarity to the action a player took we want to learn from. Meaning that given s and a, the reward is higher the more accurate[19] a resulting ball position s' is, according to the real position the player would let the ball end up in, given the same initial s and a.
- $\gamma \in [0,1]$: discount factor, which is used to model importance between present and future rewards.

In practice, we assume a finite MDP, which means that the set of states S and set of actions A are both finite, while in theory the S and A do not have to be finite [25]. Therefore the used MDP here is often referred to as *finite MDP*.

The solution to the MDP modeled optimization problem is the *optimal policy*

$$\pi : S \to A$$
$$\pi(s) = a$$

which returns a particular action to a particular state, while optimizing the overall reward at the same time, written as

$$\sum_{t=0}^{\infty} \gamma_t R(s, a, s') = \sum_{t=0}^{\infty} \gamma_t R(s_t, a, s_{t+1})$$

[17] Whereas in *continuous* time steps, there are infinitely many time steps between arbitrary points in time.
[18] According to [32], we use $Pr(A)$ to indicate the probability of an event A.
[19] Accuracy resp. the used similarity measure is described in Section 5.6 on page 24.

with $a = \pi(s)$ according to the current policy and t indicating the current time step.

One notable optimization technique for finding this policy is *dynamic programming*, invented by Richard Bellmann in the 1960's. Dynamic programming finds a solution by dividing an optimization problem into smaller sub problems, recursively optimizing and storing optimal solutions of these sub problems, which will eventually satisfy the optimal solution of the original problem [4]. One example method, which uses this principle, would be Dijkstra's Algorithm, mentioned in Section 4.1. A more general approach would be reinforcement learning, also mentioned in the related work section on page 6.

5.3.2 Representation of BalanceFit

In order to represent events in the game BalanceFit, the board angle $pos_{board} = (x, y, z)^\intercal$ and ball position $pos_{ball} = (x, y, z)^\intercal$ can be seen as a state $s \in S$. I f a state is recorded during a running game, the initial ball velocity $v_{ball} = (x, y, z)^\intercal$ is also part of the state, leading to $s = [pos_{board}, pos_{ball}, v_{ball}]$.
The action can also be represented as a vector in the euclidean space where the board is situated, because a board angle, which is different from the current one, will result in an implicit altering of pos_{ball} and v_{ball}, computed by the Unity engine $\mathscr{E}$. Therefore we can represent the action $a = (x, y, z)^\intercal = pos'_{board}$ as a vector as well.
With $s = [pos_{board}, pos_{ball}, v_{ball}]$ as the state and $a = [pos'_{board}]$ as the action, feeding state and action to the engine $\mathscr{E}$ to derive a resulting state can be seen as

$$\mathscr{E}(s_n, a_n) \rightarrow s_{n+1}$$

with $n \in \mathbb{N}$ indicating the current time step.

The error e_n in order to optimize our model is computed as the difference between the actual state s_n and the expected or desired optimal state s^*_n

$$e_N = \sum_{n=0}^{N} |s_n - s^*_n|$$

with $|*|$ as an arbitrary norm. Due to complexity reasons, we start evaluation after one full run of a game level, meaning a state can also be expressed as the set of single states during a whole run. Further the action can be seen as the current parameter set at the start of the run, causing all the single alterations in board angles during the run. In our experiments we use $s = pos_{ball}$ as the single state and for error measurement the Euclidean distance. The idea for using only the ball position as the state is is explained in Section 6.4.2, the reason for using the Euclidean distance for error measurements explained in Section 5.6 and the general experiment setup can be seen in Section 6.4.

5.4 Machine Learning and Optimization

Machine learning and optimization are strongly connected. Machine learning, as a sub area of artificial intelligence, uses mathematical optimization methods to create algorithms, which satisfy the criteria of being intelligent when finding a solution.

5.4.1 Artificial Intelligence and Machine Learning

Artificial Intelligence describes the field of creating a hard- or software, which shows intelligent behavior. The general problem is to find a solution, which accomplishes a task in an intelligent manner, often even better than the human itself in specific domains. Games are of special interest for AI researchers, because they happen to be a deterministic environment, which allow to derive solution strategies in a more convenient way while the principles and strategies often can be successfully applied to real world problems afterwards. Famous examples are the defeat of the reigning chess world champion Garry Kasparov by DeepBlue[20] in 1997, or most recently the defeat of one of the strongest players of the game of Go by AlphaGo[21] in Spring of 2016, where one of the best human players was outplayed by an artificial intelligence.

[20] Developed by IBM, mainly relying on databases and brute force computing power.

[21] Also see the related work section on page 6. Here the engine uses a highly complex state of the art neural network with multiple layers to learn playing patterns, rather than raw computational power.

Machine learning, as one of the largest subfields of artificial intelligence, can be described as recognizing patterns in data and fit a model to it in order to make predictions or decisions[7]. Here famous examples are the recognition of handwriting as an example of *supervised learning*. Examples of handwritten letters are fed to the machine learning algorithm, already assigned to their corresponding letters. The algorithm then derives a function, which takes pixels of a handwritten letter as input and returns an actual letter as output. This function can be optimized by feeding it with more test data, until it is sufficiently accurate in recognizing handwriting correctly. Learning from demonstration can also be seen as a form of supervised learning.

5.4.2 Optimization

Optimization is a field of applied mathematics, where the goal is to maximize or minimize a function, also called loss function or cost function. This function maps parameters to a real number, representing the current cost[22]. This comes in handy when solving data-driven tasks like machine learning problems. Input samples are systematically fed to a given function $f_0(x)$ in order to find a value $x_0 \in \mathbb{R}^n$, such that $f_0(x_0) \leq f_0(x)$: $\forall x \in \mathbb{R}^n$.

This is called the *optimization problem* as described in [9]:

$$\begin{aligned} \underset{x}{\text{minimize}} \quad & f_0(x) \\ \text{subject to} \quad & f_i(x) \leq b_i,\ i = 1, \ldots, m. \end{aligned}$$

with *objective function*[23] $f_0(x) : \mathbb{R}^n \to \mathbb{R}$ and *inequality constraints* $f_i(x) \leq b_i$. Optimization problems are usually stated as a minimization problem. However by simply negating the objective function, the according maximization problem can be derived.

Optimization problems often have multiple good solutions. Classical methods are highly iterative and can therefore not guarantee finding these different solutions. A popular approach to this are *evolutionary algorithms*, which are inherently suitable for finding different multiple solutions, due to their technique of altering their values fed to the function in each iteration.

5.5 Evolutionary Algorithms

Evolutionary algorithms (*EA*) solve optimization problems by adopting principles of biological evolution. In nature, every individual has a different appearance called *phenotype*, which is determined by its *genotype*, the unique genetic sequence. *Natural selection*, the key mechanism of evolution, states that individuals differ in surviving and reproducing due to their different phenotypes. The better adjusted to their environment, individuals are more likely to survive and reproduce, which results in combining the underlying genetic material of their phenotypes. Reproduction of individuals of different phenotypes combines different genotypes, hence the genotype is a passive data structure, implicitly determined by the phenotype, which is directly succumbed to natural selection [8].

Figure 5 shows an evolutionary algorithm in principle, resembling the principles in combining offspring in a different way than the parental generation and select the individuals with the highest fitness value for the next generation, as shown in [37]:

1. Starting point is some initial population.
2. Individuals experience change in their chromosomes (*mutation*).
3. Individuals will be combined with the rest of the population again (*recombination/crossover*).
4. A subset of the population is selected for evaluation (*selection*).
5. The individuals of the subset are fed to the fitness function (*evaluation*).
6. Individuals with the best fitness values survive.
7. This process is looped until convergence.

[22] Often also called loss or error.

[23] The cost function or, in the case of evolutionary algorithms, *fitness function*.

The result is then an optimized population satisfying the criteria of the fitness function.

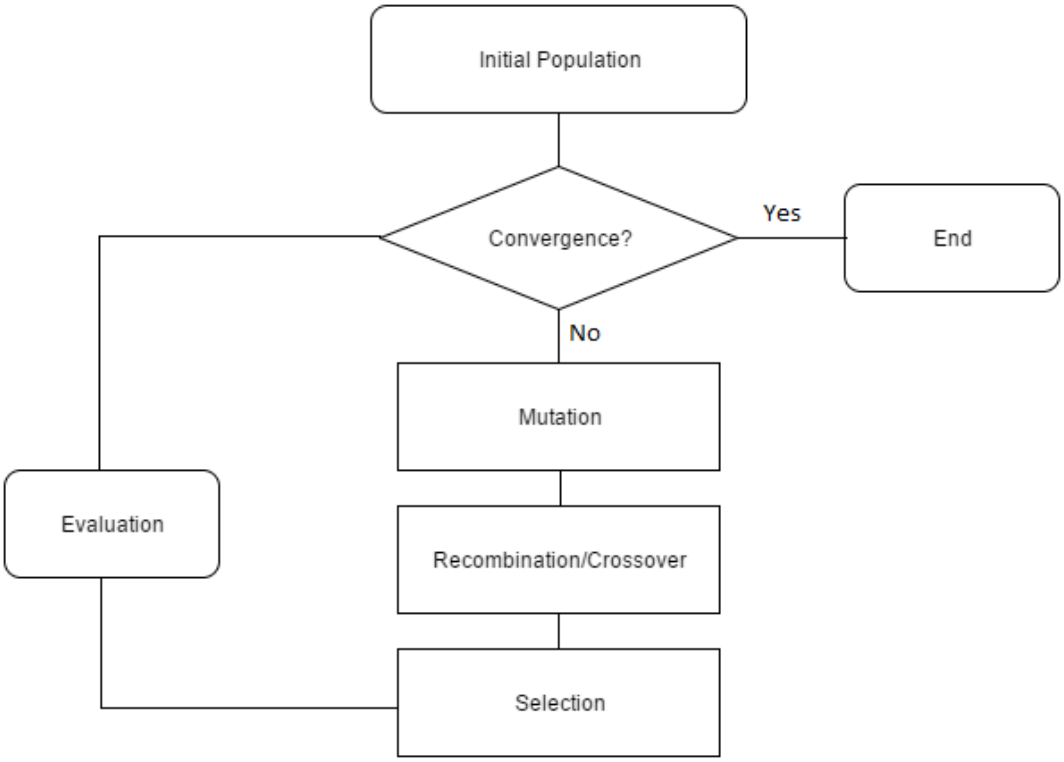

Figure 5: The process of an evolutionary algorithm.

5.5.1 Fitness Function

The fitness function is the heart of the optimization process and induces the process of selection by evaluating the different individuals in different generations:

$$f : \mathbb{R}^n \to \mathbb{R}$$

$$f(x) = \sum_{i=1}^{n} e_i = \sum_{i=1}^{n} |x_i^n - x_i^{n*}|$$

which is basically the sum of errors of the current n individuals x and the optimal solution x^* we are trying to achieve during this optimization process. An individual x can also be seen as the independent variables of a parametric function. These parameters are being optimized with respect to a goal, usually minimizing an error, resp. the value of an error function[24]. Multiple or even contradictory goals, represented by several fitness functions, can be modeled as the weighted sum of the single fitness functions:

$$F(x) = \sum_{i=1}^{n} w_i f_i(x)$$

with $\sum_{i=1}^{n} w_i = 1$.

[24] See Section 5.4.2

5.5.2 Genetic Operators and Elite Strategy

The different individuals are created using the principles of *mutation* and *crossover* to direct the algorithm towards a solution during the optimization process. Existing solutions, called *chromosomes*, are combined with slightly changed chromosomes (mutation) to a new set of solutions (crossover) to evaluate them to form the next generation of chromosomes (selection). These three operations (mutation, crossover and selection) are called *genetic operators*.

- **Mutation:**

  As seen in Figure 6, mutation of a bit string can be achieved by changing a bit in a given string of length n, with mutation probability $\frac{1}{n}$, for each bit.

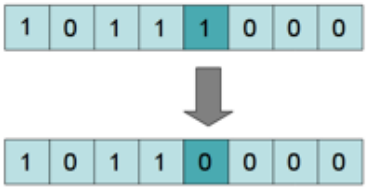

Figure 6: Flipping a single bit as mutation.

  Instead of bit flipping, for integer or float values, the value is randomly replaced by a number $n \in [a, b]$ between the lower bound a and upper bound b, called *boundary mutation*. Besides bit flipping and randomly alter chromosomes in boundary mutation, other mutation strategies are commonly used:

  - **Boundary mutation**: as mentioned above, randomly assigns a value $n \in [a, b]$ between the lower bound a and upper bound b.
  - **Gaussian mutation**: $n \in [a, b]$ is not randomly chosen, but Gaussian-distributed.
  - **Uniform/ non-uniform mutation**: $n \in [a, b]$ is a uniform random value resp. a non-uniform random value. The latter can help accelerating the mutation rate.

- **Crossover:** Crossover varies the chromosomes by combining the chromosomes. This can be done using several ways, the most common is the the one point crossover, shown in Figure 7.

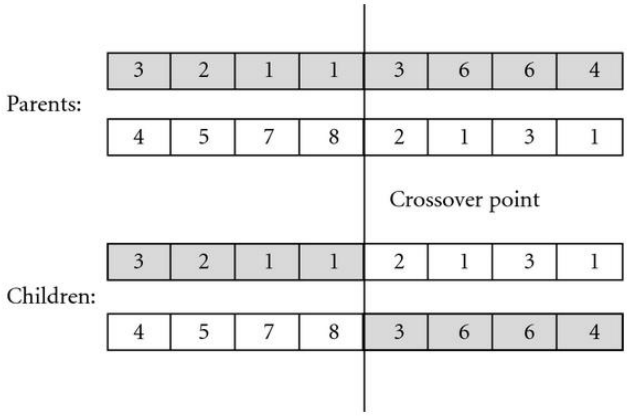

Figure 7: Crossover with a single crossover point.

  Having several crossover points is also common, as seen in Figure 8 depicting a two-point crossover.

  However these crossovers are consistent in length of the offspring, meaning that the length of the resulting chromosome matches the parental chromosome length.

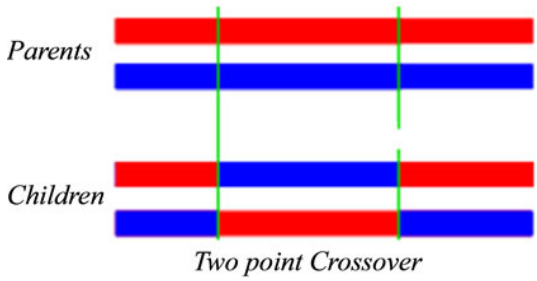

Figure 8: The two-point crossover

- **Selection:** After mutation and recombination is applied to the given population, selection is done in order to form a new population X_{new} with n individuals out of an existing population X_{old}, using the following steps for a fitness function $f : \mathbb{R}^n \to \mathbb{R}$ and $i, j \in \mathbb{N}$:

  1. The fitness function is evaluated for each individual x_i.
  2. The population is sorted according to their fitness values, usually in an ascending order.
  3. Normalization of the population with respect to their fitness values, such that $\sum_{i=1}^{n} f(x_i) = 1$ for each individual x_i.
  4. Add x_i to X_{new} if $f(x_j) \leq f(x_i) \leq r : \forall x_j \in X_{old}, r \in [0, 1]$, meaning that the first individual is chosen, which exceeds the threshold r, which is a randomly chosen number between 0 and 1.
  5. These steps are looped until $|X_{new}| = n$, meaning n individuals were found for the new population.

- **Elite Strategy:** Elite strategy allows a percentage of the best solutions to stay within the current population, by letting it pass through to the next generation without being modified. A typical value lies between 5 % and 10% [15]. For example, using elite strategy with a percentage of 8% in a population of 100 individuals would mean that the top 8 individuals will be transferred to the next generation without alteration at all. The point here is that any future population is at least as good as the current population if elite strategy has been employed.

  If elite strategy is not employed, it is often advised to store the best individuals as an external solution at least, in order to ensure a better evaluation and to not lose earlier local optimal solutions [8]. Storing elite solution is also needed for some termination criteria[25]. However, using elite strategy can result in duplicating best individuals, which should be considered and therefore solutions in every generation should be checked for duplicates to avoid accumulating elite solutions exclusively.

The following example illustrates how parameters of a cubic function can be optimized to approximate a given set of points using evolutionary optimization:

***Example**:*

Given n sampling points $(x_1, y_1), ..., (x_n, y_n)$ satisfying an unknown cubic function

$$y(x) = ax^3 + bx^2 + cx + d$$

with $a, b, c, d, x \in \mathbb{R}$, where we want to find the optimal set of parameters a, b, c, d. Encoded as the chromosome $c_i = (a, b, c, d)$. The error e_i of sampling point x_i to y_i can be computed as

$$e_i = |y_{c_i}(x_i) - (y_i)|$$

[25] See Section 5.5.5 on page 23

such that the overall fitness of the chromosome c_i can be computed as $e(c_i) = \sum_{i=1}^{n} e_i$. To satisfy the condition that a low error results in a high fitness, $fitness(c_i)$ is defined as

$$fitness(c_i) = \frac{1}{e(c_i)}$$

Applying the genetic operators described in Section 5.5.2, the chromosome $c_i = (a, b, c, d)$ will eventually converge to values a, b, c, d, satisfying the proposed function.

Employing these strategies, several branches of evolutionary algorithms arose since the 1960's, which differ in the underlying data structure the individuals are represented by, depicted in table 1. We will have a deeper look into *evolution strategies*, since we are inclined to optimize real valued vectors for optimizing the BalanceFit artificial intelligence.

Year	Inventor	Method	Individuals
1965	Fogel, Owens and Walsh	Evolutionary Programming (EP)	Automata
1965	Rechenberg, Schwefel	Evolution Strategies (ES)	Real valued vectors
1975	Holland	Genetic Algorithms (GA)	Bitvectors of constant length
1992	Koza	Genetic Programming (GP)	Trees

Table 1: Trends in evolutionary algorithms

5.5.3 ES (Evolution Strategy)

In the late 1960's bionic and cybernetic researches in Berlin, Germany began developing a simple ruled strategy when it comes to designing and evaluating consecutive experiments [5]:

1. Change all variables at a time, mostly slightly at random.
2. If the new set of variables does not diminish the goodness of the device, keep it, otherwise return the old status.

These rules resembles the reproduction of individuals describes by evolutionary biology. The first one can be seen as natural mutation, the second one as natural selection or „the survival of the fittest" [40].

In evolution strategy, selection is based around parameters μ and λ, which control the *selection intensity*. Generally a set of descendants of size λ is selected out of μ parents, while μ individuals will be taken over into the new generation of size μ. The new generation is either constructed from parents and offspring (*plus-strategy*), or only out of the offspring (*comma-strategy*). From this, four major strategies can be distinguished:

- $(1+1)$**-ES**
  Also called *two-membered ES*, is the most basic ES, first investigated in [36]. Here, there is 1 parent and 1 offspring individual, meaning a population of size 2. If the offspring's fitness value is at least as high as the parent's, it will be the parent of the next generation.
- $(\mu+1)$**-ES**
  Also called *steady-state-ES*, is a *multimembered ES*, with μ parents at a time. Two of them are chosen at random and recombined to form an offspring individual. If this individual has a fitness value higher than any of the parent individuals, it is taken over into the next generation, else it is discarded. This will eventually discard all individuals with the lowest fitness value of either parent or offspring generation.
- $(\mu+\lambda)$**-ES**
  Similar to the previous $(\mu+1)$-strategy, but here, besides μ parents at a time, also $\lambda \geq 1$ descendants are created.

λ individuals out of all the $(\mu + \lambda)$ individuals are discarded with respect to their fitness values before forming a new generation of size μ.

- (μ, λ)-**ES**
  Here selection of the new population happens only among the λ offspring, while the parental population μ is fully discarded. This strategy only works if $\lambda > \mu$ holds, meaning this strategy lives off from a birth surplus.

Except for the (μ, λ)-ES, all strategies keep the population size constant.

5.5.4 Exploitation vs. Exploration

When searching for a solution in the parameter space, one can try to improve an existing and promising solution by looking for further improvement in the region around this solution. This would be *exploitation*, because one tries to exploit the existing solution by looking for better optimum in the region around it. This is also refereed to as *local search*.

Since one cannot say for sure if this is just a local or already a global optimum, the former approach would be to discard this solution and look for another potential optimum in a different region of the solution space, called *exploration*. This avoids being trapped in a local region, therefore called *global search*, but also cannot guarantee that the new optimum will be better than previous optimum already was, which already describes the *exploitation-exploration trade-off* as a general dilemma in machine learning optimization.

In the context of evolutionary algorithms, exploitation is done via selection, while exploration is done via the genetic operators, crossover and mutation [17]. This means that without mutation, one could only rely on selection, therefore exploiting the given population. Without selection one would employ pure exploration by constantly changing regions in the chromosome space and never tending towards a possible solution. The optimal strategy to tackle this problem is always dependent on the data and the goal one wants to achieve. One way of dealing with this is by controlling the selection intensity using the parameters μ and λ in the presented evolution strategies described in 5.5.3.

We make use of the capability of evolutionary optimization in exploring new regions in the search domain, even when being stuck in a local region for some generations. We are dealing with a non-convex optimization problem, where a local optimum does not necessarily have to also have a global optimum. Since no solution or existence of a solution for parameter settings of the BalanceFit AI for approximating human player behavior is known, evolutionary optimization is useful for searching within this non-convex domain.

Converging to a local minimum due to lack of exploration is an issue more recent ES-variants are trying to tackle using a *multipopulation approach* known as $(\mu/\rho + \lambda)$-ES, where the λ parents are recombined in groups of ρ individual to generate λ descendants.

5.5.5 Termination and Code Examples

A *termination function* has to be defined in order to stop the optimization process at a defined point during the optimization process. This point or the *termination condition* can be divided into three groups according to [24]:

- **Direct Termination Criteria:**
  - Maximal number of generations or time:
    Either a set number of generations is reached or a given time budget is consumed.
  - A solution is found or lies within a bound:
    A set of parameters is found which satisfies the objective function or a bounded error respectively.
- **Derived Termination Criteria:**
  - Moving average/ running mean:
    Difference between current error e_n and mean of error of last n generations $\sum_{n=t}^{N} e_{N-t}$ is less than a given threshold.
  - Standard deviation:
    The error deviation of the last n generations is withing a given bound.

- Best-Worst:
  Then error between the best and the worst generation is less than a given threshold.

- **Cluster-Based Termination Criteria:** The basic idea here is that the distribution of individuals in the parameter space with respect to their fitness is examined. For example a termination would be caused when the individuals do not leave a certain cluster with more optimization steps. Further details are given in [24].

While the direct termination criteria avoid premature termination[26], the intuitive idea behind the derived termination criteria such as standard deviation or the best-worst termination criterion, is that the current best solutions reached a plateau, so that more iterations would not lead to better results and therefore reduce needless computation. Combinations of multiple termination criteria are also possible.

5.6 Evaluation

The evaluation of our optimization process is centered around the objective function[27], or fitness function in case of evolutionary optimization. We evaluate our model by evaluating the fitness function, because this allows us to assess the parameters altered by the optimization. The evaluation yields the error of our current model. The optimization is designed to modify parameters of the model in order to decrease the error in the next step and ultimately minimizing the error.

For comparison of two runs, the distance of the ball position $pos^t_{ball} = s_t$ at time step t can be computed with respect to the Euclidean distance, which is computed as:

$$dist(s_t, s^*_t) = ||(x,y,z)^\intercal_t - (x,y,z)^{\intercal *}_t|| = \sqrt{(x_t - x^*_t)^2 + (y_t - y^*_t)^2 + (z_t - z^*_t)^2}$$

where s_t is the current state and s^*_t the desired state from the initial run at time step t.

Another measure of similarity would be the cosine similarity and is computed as:

$$sim(s_t, s^*_t) = cos(\theta) = \frac{s_t \cdot s^*_t}{||s_t||\ ||s^*_t||} = \frac{(x,y,z)^\intercal_t \cdot (x,y,z)^{\intercal *}_t}{||(x,y,z)^\intercal_t||\ ||(x,y,z)^{\intercal *}_t||}$$

where $s_t \cdot s^*_t$ is the dot product between vector $(x,y,z)^\intercal_t$ and $(x,y,z)^{\intercal *}_t$ given as

$$\sum s_t s^*_t = x_t x^*_t + y_t y^*_t + z_t z^*_t$$

and $||s_t||$ and $||s^*_t||$are given as

$$||s_t|| = ||(x,y,z)^\intercal_t|| = \sqrt{x^2_t + y^2_t + z^2_t}$$

$$||s^*_t|| = ||(x,y,z)^{\intercal *}_t|| = \sqrt{x^{*2}_t + y^{*2}_t + z^{*2}_t}$$

The cosine similarity is measured by the cosine of the angle between two vectors, while the Euclidean distance computes the direct distance in the vector space, as seen in Figure 9. The similarity value ranges from -1 to 1, where 1 indicates identical, 0 orthogonal and -1 exactly opposite vectors.

[26] Termination before reaching an optimum.

[27] Or evaluation function, see Section 5.4.2 on page 18.

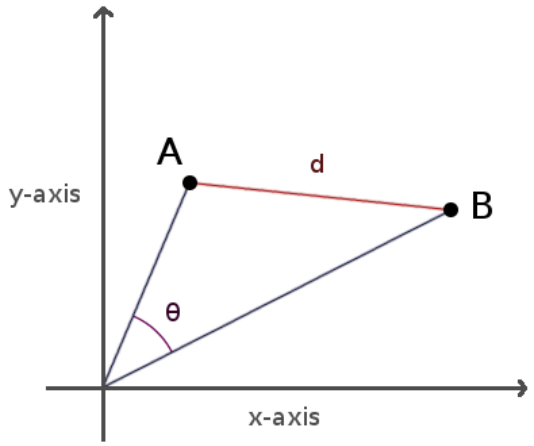

Figure 9: Difference between Euclidean distance d as a direct distance between vectors A and B, and cosine similarity ϕ as the angle between vectors A and B.

The game fetches the states every 0.1 seconds. In order to minimize the effect of the random noise of the physics engine, the error estimation usually averaged over 10 single runs with the same parameters before compared to the sample run. The error of a run is estimated as

$$\frac{1}{T}\sum_{t=1}^{T} dist(s_t, s_t^*) = \frac{1}{T}[dist(s_0, s_0^*) + dist(s_1, s_1^*) + ... + dist(s_t, s_t^*)]$$

which is the sum of the distance between the current state and the desired state at time step t, normalized with respect to the number of runs. Similarly the evaluation of a run can be computed using the cosine similarity as

$$\frac{1}{T}\sum_{t=1}^{T} sim(s_t, s_t^*)$$

6 Experiment Setup

This chapter describes the experimental setup and the data used for the experiments.

6.1 Dataset Structure

An initial data set is collected recording a sample run. For the first series of experiments, the sample run stores only the ball position pos_{ball} along with the corresponding time steps. The intuitive idea here is that pos_{ball} corresponds to the path the sample run was following, meaning that if a parameter set is found, which approximates pos_{ball} for every time step for the sample run to a certain error, the playing behavior is sufficiently learned with respect to the path through the labyrinth.

However, only following the ball path does not necessarily simulate the behavior of the player shifting their center of mass. Therefore additionally to pos_{ball}, taking the board angle pos_{board} into account would be more suitable for comparing forces applied to different regions of the balance board when leading the ball through the labyrinth, which is subject of the second series of experiments.

6.2 The Genetic Algorithm Framework

As an optimization environment the Genetic Algorithm Framework for .NET (GAF) [31] was used, which includes all the genetic operators discussed in Section 5.5.2. During optimization single and double point crossover, mutation, random replace and elite strategy was employed.

The parameters are set as follows:

- Number of chromosomes: 10
- Crossover Probability: 65%
- Mutation Probability: 20%
- Elite Percentage: 10%

Where a chromosome indicates one possible solution and elite percentage indicates, that 5% of the best solutions are preserved until the next generation.

The chromosome length is determined by the number of parameters to be optimized. Also the PID-controller parameters are changed by 0.5 per generation and the non-Boolean wall and pathfinding parameters are changed by 0.1 per generation. The Boolean parameters are either $true$ or $false$ per generation. A direct termination criterion is an error threshold of 0.04, derived in Section 6.4.1.

6.3 Relevant Parameters

Crucial parameters are both PID-gains of the board and balance controller and therefore almost always subject to the optimization process. We will successively add more parameters respectively test the Boolean parameters like wall factor and other pathfinding parameters[28]. The parameters of the experiments are listed below, the other parameters remain as default values:

Tilt controller parameters:

- K_{P-NC}
- K_{I-NC}
- K_{D-NC}

Balance controller parameters:

[28] For a reference overview over all the control parameters and their role, see Section 4.2.3 on page 13.

- K_{P-BC}
- K_{I-BC}
- K_{D-BC}

Pathfinding Boolean parameters:

- $useDiagonals$
- $useOuterDiagonals$
- $useWalls$

Pathfinding continuous parameters:

- $leaningAngle$
- $wallfactor$
- $holeRadius$

6.4 General Setup

For comparing different runs with different parameter settings, a reference run is recorded as the original sample. A run is a record of states while playing a level. A different run can be compared to the reference run by computing the distance between the states, e. g. by computing the distance between a state in the initial run and state in the test run at time t for every time step by using the Euclidean distance or cosine similarity. All experiments were done in wood level 1. The hole parameter experiment was conducted in wood level 6.

The AI sample runs are initialized with *optimal parameters*

$$par^*_{NC} = (K_{P-NC}, K_{I-NC}, K_{D-NC}) = (5, 3, 0)$$

for the board angle controller and

$$par^*_{BC} = (K_{P-BC}, K_{I-BC}, K_{D-BC}) = (20, 15, 0)$$

for the balance controller. The AI the starts optimizing with *initial parameter* settings

$$par^0_{NC} = (K_{P-NC}, K_{I-NC}, K_{D-NC}) = (1, 1, 0)$$

and

$$par^0_{BC} = (K_{P-BC}, K_{I-BC}, K_{D-BC}) = (1, 1, 0)$$

For the *non-optimal parameters* for the controller values in Section 6.5.1 and 6.5.2, the tilt controller is set to

$$par_{NC} = (K_{P-NC}, K_{I-NC}, K_{D-NC}) = (10, 5, 1)$$

and the balance controller to

$$par^*_{BC} = (K_{P-BC}, K_{I-BC}, K_{D-BC}) = (15, 10, 1)$$

This is a non-optimal setting and there is room for optimization, but still shows reasonable behavior at the same time. All gain values settings are also inspired by domain experts who are experienced with the BlanceFit AI.

6.4.1 Difference Between Euclidean Distance and Cosine Similarity

Due to the non-deterministic simulation of physical properties of the underlying Unity engine, a run done by the AI is never measured with an error of zero, even when using the same initial parameter settings. Figure 10 shows 50 runs while computing the error of the ball position pos_{ball} of the current run with an initial run, using the same set of parameters.

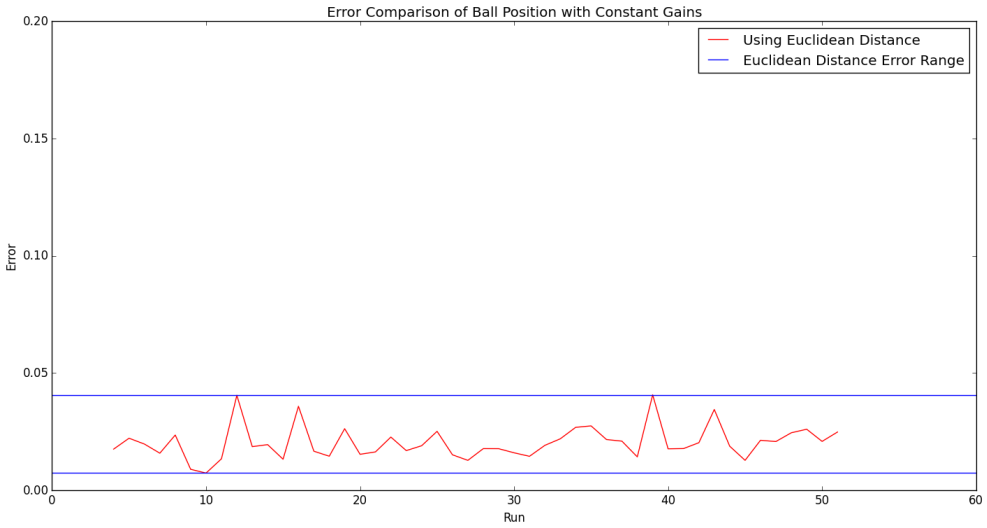

Figure 10: Error results of 50 runs with constant gain parameters using Euclidean Distance. The lowest error is 0.00736 and the highest error 0.04072, resulting in a range of 0.03336, also depicted by the blue parallel lines. The error was measured between the ball positions using the Euclidean distance.

It can be seen that there is still a deviation from zero, even when using the exact same settings and the same starting position in the same level for a comparison run. Therefore when using error measurement for convergence, it has to be taken into account, that termination can also be achieved through a direct termination criterion[29] when the error is within an interval of slight change, e.g. in $[0.007, 0.040]$ taking on values in a range of $\epsilon_{Euclid} = 0.033$ like in the above example.

When conducting the same experiment with the cosine similarity as the error measure, similar behavior is shown. The range of the cosine similarity lies within $[-1, 1]$, the optimal value would be 1 and indicates the highest similarity. To convert the similarity value to a distance value, the plot shows $1 - sim(s, s^*)$, shifting the range of the similarity measure to $[0, 2]$, where 0 would now indicate the mot optimal value, or lowest error, similar to the range of the Euclidean distance.

This value is also never met, but the range in which the similarity value lies is $[0.003, 0.096]$, resulting in a higher $\epsilon_{cosine} = 0.092$.

[29] On termination criteria see Section 5.5.5 on page 23.

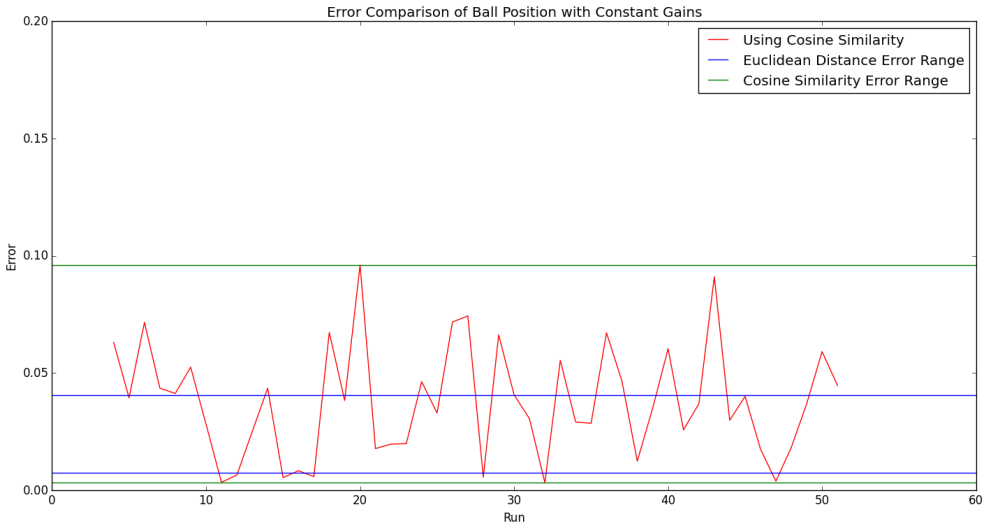

Figure 11: Error results of 50 runs with constant gain parameters when using cosine similarity . The lowest error is 0.0032 and the highest error 0.0960, resulting in a range of 0.0928.

The non-deterministic behavior of the error still allows converging to an optimum when comparing runs with different parameter settings. Figure 12 shows the constant run error in red as a baseline in comparison to runs with different parameters. Alternating the P- and D-gain of the board angle controller only by $+/-1$ per generation, thus approaching the initial values and having the error converging towards the given boundary after 16 runs.

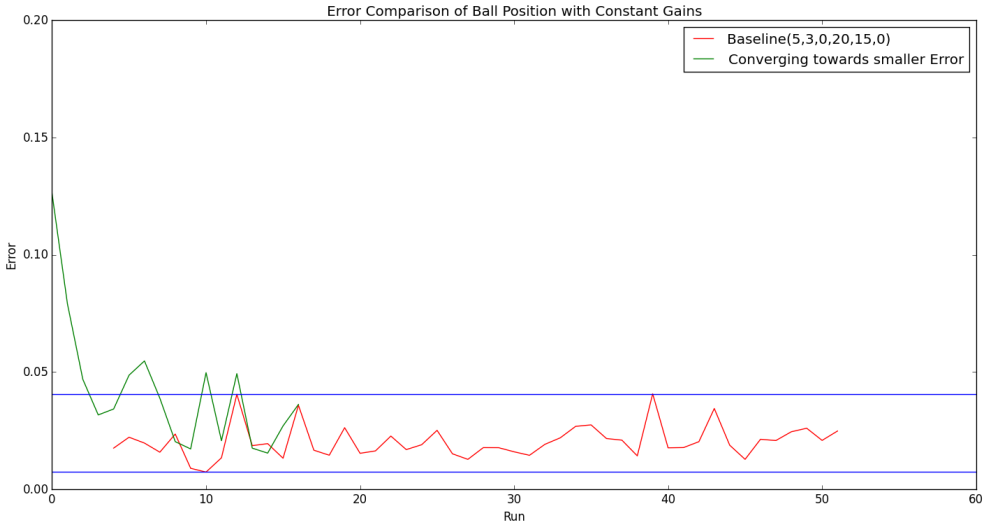

Figure 12: Convergence of the error towards the error threshold when optimizing D- and P-gain of the board angle controller. The constant error baseline from Figure 10 is given as well.

The Euclidean distance shows less deviation in estimating the error and is therefore used in the evaluations unless mentioned otherwise. A baseline as the maximum error when using ball position for error estimation is set as a direct criterion to an error of 0.04 and indicated by the blue parallel line in the plots. As soon as the learning curve reaches as an error less or equal than 0.04, we can assume that we approximated the sample run sufficiently.

6.4.2 Difference Between Ball Position and Board Angle for Error Estimation

Here the difference between the ball position and the board angle is shown. While the ball position provides a reliable estimation of an error and is constant within the evaluation error range ϵ, the board angle as an error criterion does not show such behavior. Figure 13 shows 50 runs with constant parameters and the board angle as an error measure. The minimal error is 0.00811, the maximum error is 0.19175, resulting in a range of 0.18363.

In comparison to the ball position as an error estimate, here the error is always higher. The maximum error of 0.19175 exceeds the maximum error when using the ball position by 0.15, which is already 4.5 times more than the whole error range of the ball position error, which is only 0.03336.

One reason can be a bias when fetching the board angle every 0.1 seconds, meaning that the board angle changes more often than the ball position and causes bad angles during fetch time. Another reason could be, that the board angle is not as much of a factor as the ball position, when trying to have a similar run. But in both cases the board angle is not reliable in deriving to a direct convergence criterion and is therefore neglected. The ball position is used for estimating the error during the optimization instead.

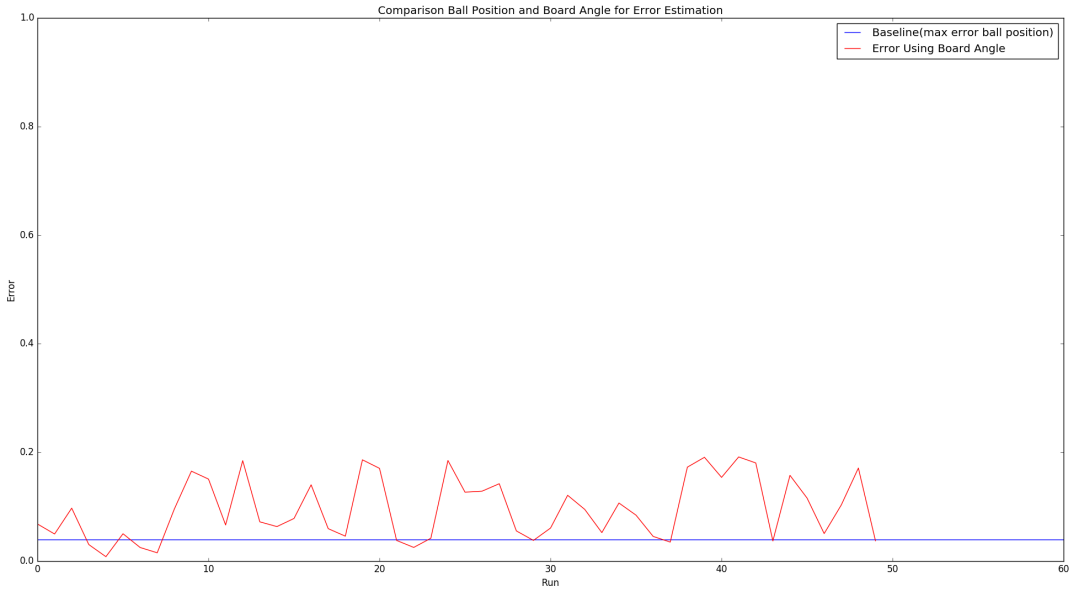

Figure 13: Error results of 50 runs with constant gain parameters using the board angle for error estimation. The lowest error is 0.00811 and the highest error 0.19175, resulting in a range of 0.18363. The blue baseline is the maximum error when using ball position for error estimation at 0.04.

6.5 Experiment Series 1: Letting the AI Learn Its Own Behavior

The first series of experiments is aimed at letting the AI learn its own initial values. In order to do this, a sample run is played by the AI with a set of optimal parameters par^*. Another instance of the AI is initialized with a different setting of parameters. Different experiments regarding the main parameters of the AI are conducted in the following sections.

6.5.1 Tilt Control: Board Angle PID-Gains

The first experiment optimizes the P-, I- and D-Gain of the board angle controller. Apart from the the PID-controller parameters for tilt control, all the other parameters are not affected by the optimization process and remain optimal. After starting the optimization with initial settings, The tilt control learning process finishes after 28 generations with an error of 0.0293.

The run with non-optimal balance controller started at 0.1924 and finished with an error of 0.0787 after 26 generations. Averaging for the states is done over 5 runs.

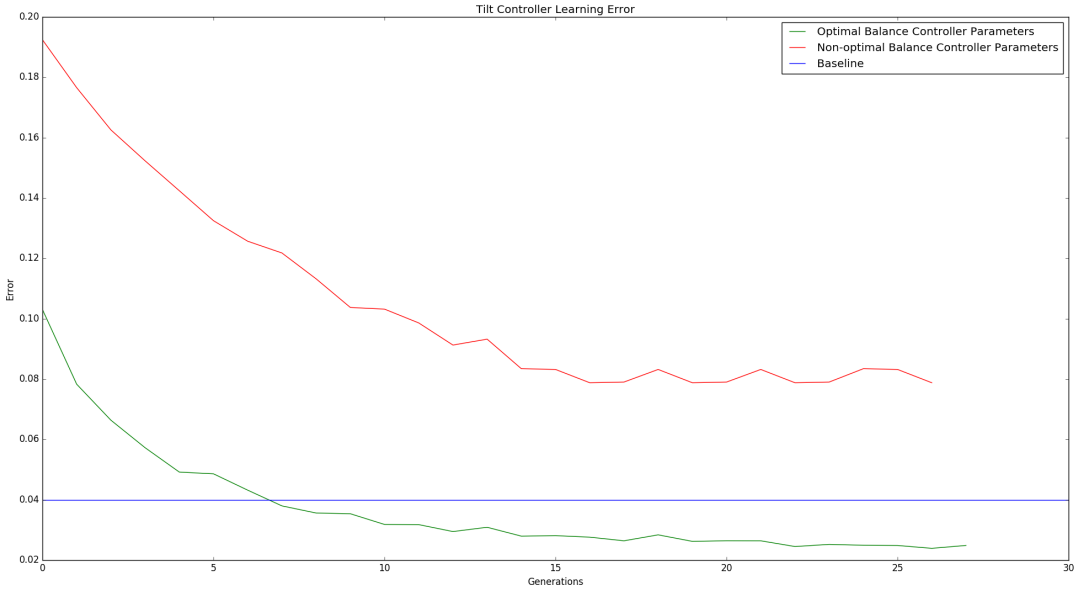

Figure 14: Tilt control parameter approximation finishing after 28 generations with a final error of 0.0293 with optimal balance controller parameters indicated by the green line. The red run with non-optimal balance control parameters finished after 26 generations with a final error of 0.0787.

6.5.2 Balance Control: Center of Mass Control PID-Gains

The second experiment optimizes the P-, I- and D-Gain of the balance controller. The tilt controller gains are not affected in order investigate the difference in impact both controller shave on the learning process. The balance control learning process with optimal tilt controller finishes after 24 generations with an error of 0.0286. The run with non-optimal tilt controller finishes after 23 generations with an error of 0.2231, starting from 0.46737.

When only optimizing the balance control, it can be seen that this controller has not as much influence on the overall quality of the approximation as the tilt control. While the tilt control can compensate for non-optimal balance parameters[30], the balance control parameters can only reduce the error to a satisfying value when the tilt control parameters are already optimal.

[30] See Figure 14.

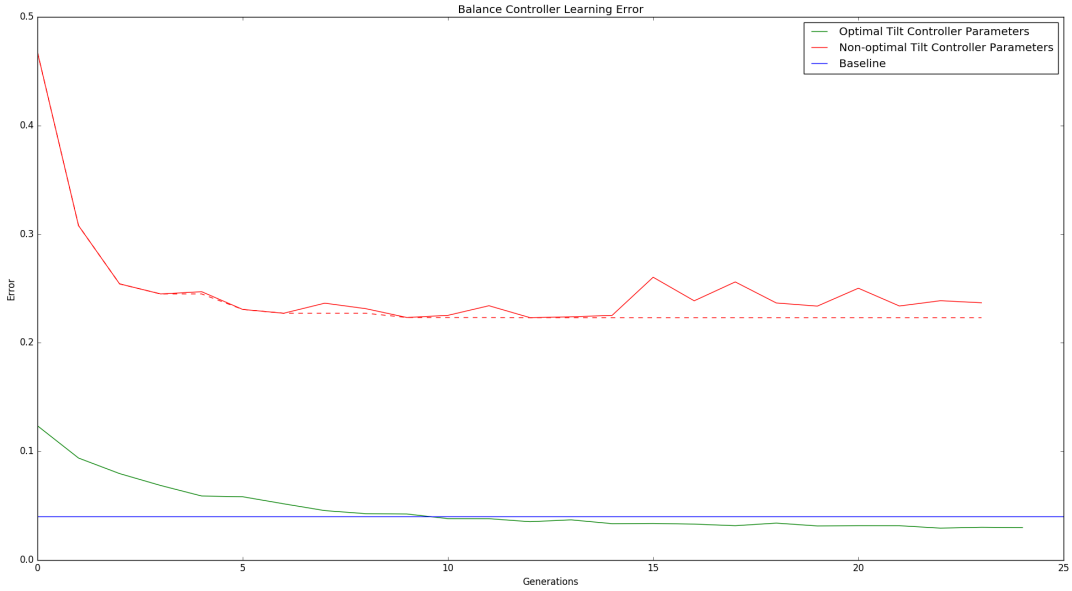

Figure 15: Balance control parameters are not as influential as the tilt control parameters for the overall approximation. The green run indicates an optimization of the balance controller with already optimal tilt controller parameters, resulting in an error of 0.0286 at convergence after 24 generations. The red run indicates an optimization of the balance controller with non-optimal tilt controller parameters, resulting in an error of at least 0.2231. The dashed line indicates the current best solution for each generation

6.5.3 Combining Tilt and Balance Control Parameters

When tilt and balance controls are altered at the same time, the approximation needs more iterations. Convergence with a final error of 0.02431 was reached after 39 generations. We are optimizing 6 continuous parameters right now. The states are still averaged over 5 runs.

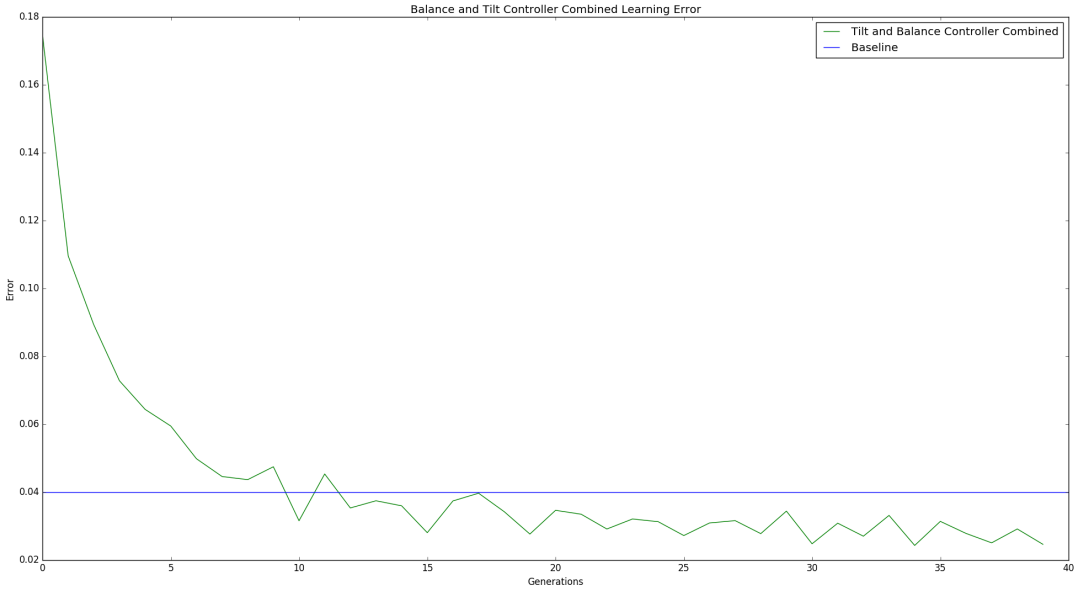

Figure 16: Optimizing tilt and balance control parameters at the same time leads to a better approximation, but needs also more iterations for convergence.

6.5.4 Role of the D-Term

When only optimizing the I-Term, no improvement in the approximation can be seen. As mentioned in Section 4.2.1, the D-term only corrects the P- and I-term error, but cannot bring the board into a position itself. Figure 17 shows this when trying to find a solution with P- and I-gains, that cause a certain error in the approximation. Two runs in Figure 17 are done with non-optimal tilt and non-optimal balance parameters respectively, while only improving the D-term. The error is not significantly reduced in both runs.

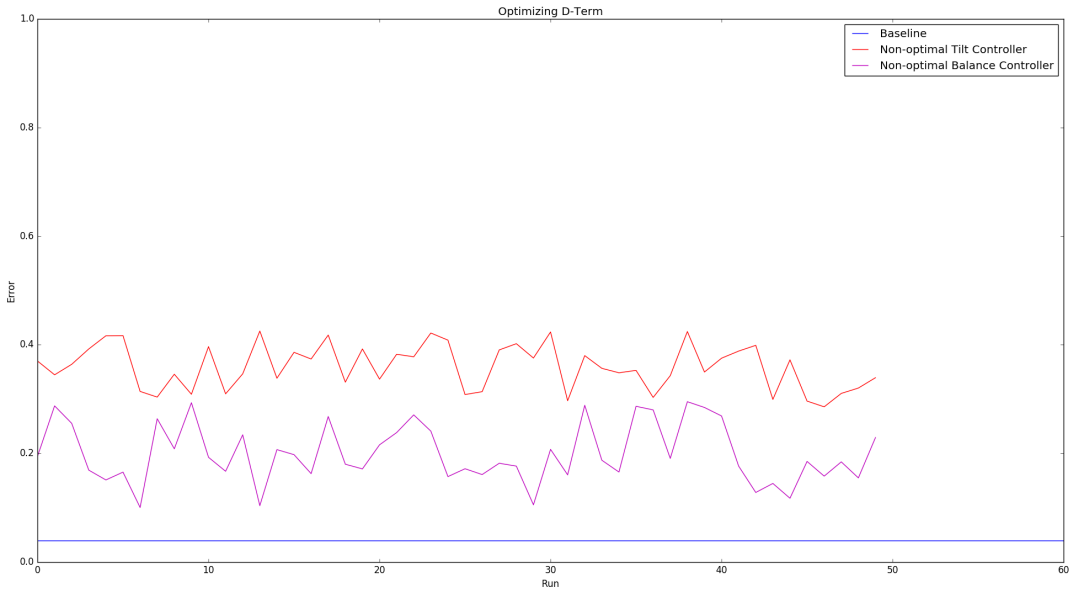

Figure 17: The corrective D-term cannot improve the solution without the P- and I-terms.

6.5.5 Role of Wall Usage

The wall usage flag $useWalls$ as part of the tilt control allows considering walls when planning a path through the labyrinth. When this flag is set to $false$, the planned path avoids touching walls. On the other hand, if the usage of walls is allowed and at the same time uses a high wall factor, the player sticks to the walls as much as possible. The $wallFactor$ and $leaningAngle$ were not part of the optimization and remained the same in all runs to only investigate the influence of the $useWalls$ flag.

Figure 18 shows two optimization runs with tilt and balance controller optimization. The sample run was initialized with $useWalls = true$. The approximation with wall usage started with an error of 0.16567 and converged after 38 generations with an error of 0.02234. In contrast to the previous experiment every state was now averaged over 10 runs, which can also be seen by the smoother learning curve.

The run without the usage of walls managed to improve the error from 0.23368 to 0.11158, but was aborted after 24 generations, because the error could not be improved after generation 15.

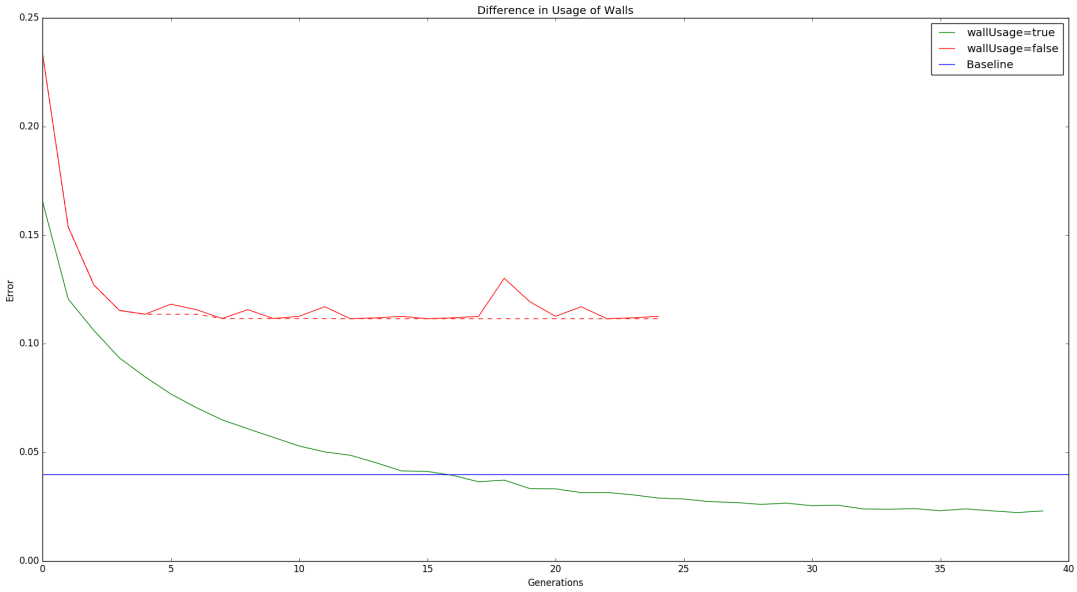

Figure 18: It is not possible to find a solution for a player, who favors walls, when learning without the usage of walls. The dashed line indicates the current best solution for each generation.

6.5.6 Including Pathfinding Parameters

Similar to the wall factor, the other pathfinding parameters might also responsible for setting the solution to an overall better approximation. For example, can a player who plans a way through the labyrinth with the outer diagonals, and thus being able to take turns in 25° and 65° direction, not exactly be copied when we learn without outer diagonals ourselves and only have 90° turns available for planning our path. This also holds for the basic diagonal parameter of 45° and for the parameters for taking holes into account while planning the path. The diagonal parameters are Boolean parameters, meaning that they are either used or not used, while the hole radius parameter are continuous values.

Allowing the hole parameters in combination with the tilt and balance controller parameters, we are already optimizing 9 continuous and 3 Boolean variables, which leads to a more complex optimization process. This is indicated by the green run and converges after 126 generations with a final error of 0.0377.

In case of learning without any diagonals, indicated by the red run, the approximation reaches at best an error of 0.1021 and was aborted after 29 generations, when after generation 21 the error started increasing again. When allowing diagonals of 45°, indicated by the purple run, but not the outer diagonals, the error is first decreasing, but does not improve after 160 generations with an error of at least 0.0627. Averaging is still done over 10 runs per state.

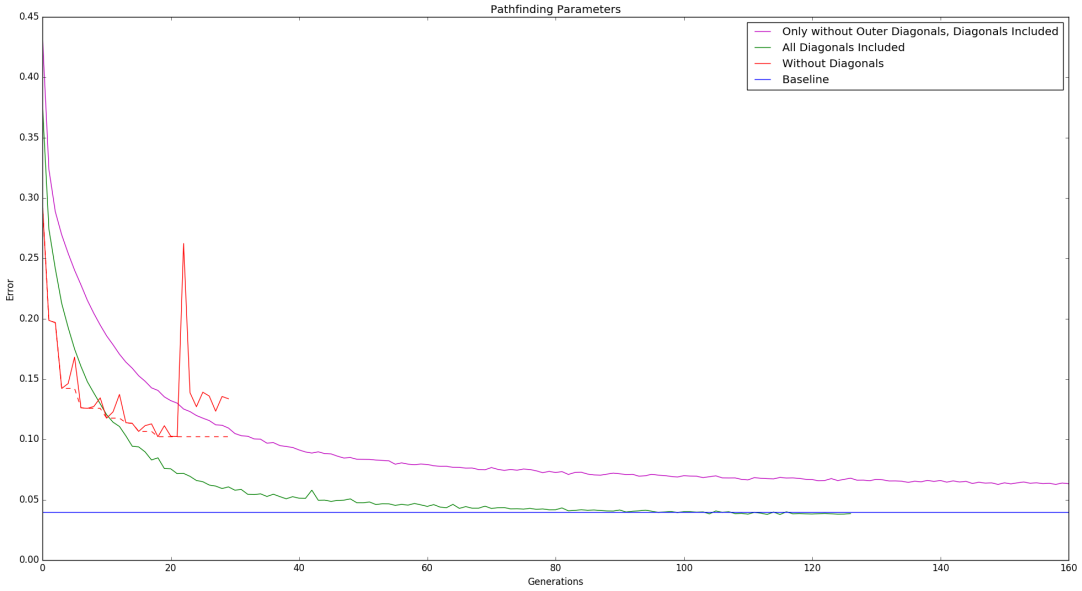

Figure 19: Taking the pathfinding parameters into account causes a longer learning process. Also without the usage of diagonals an approximation is not possible, while using only $45°$ can improve the approximation to a an error of 0.0627. The dashed line indicates the current best solution for each generation.

6.6 Experiment series 2: Learning of Human Player Behavior

When evaluating runs played by the AI, the error should not be measured right from the first state on, because the AI immediately starts alternating the board after the level started. A human player however needs some time to react to the start of the level. Therefore the human player data is cleaned before the AI learns from it and states are dismissed during the first moments of the level, where no action is taking place.

As a training set, pre-recorded sample runs of the wood world were used, named *random human run*. Among these runs, the best performing one was chosen for the plots. However, these runs were not applicable for learning as seen in the first experiments. As a result special wall-affine runs were recorded with the goal trying to copy an AI run with wall usage. The idea is to enforce a more successful learning process and have a more heterogeneous training set, because the untrained applicants of the pre-recorded runs showed often times no target-oriented behavior due to the lack of experience in playing the BalanceFit game. The wall-affine runs, referred to as the *wall run*, was done in the wood world level 1 and in wood world level 6 in order to use it for the hole parameter experiments as well. These specific experiments are discussed in Section 6.6.4 and Section 6.6.5.

6.6.1 Tilt Control: Board Angle PID-Gains

Like in the AI experiment series, the first experiment uses the P-,I- and D-Gain of the board angle controller. The control parameters for the board tilt are influencing the approximation, but cannot approach a solution with an error lower than 0.71059 in random human runs, which was reached after 39 generations and is depicted by the purple line. For the green wall run after 53 generations an error lower than 0.20849 could not be reached.

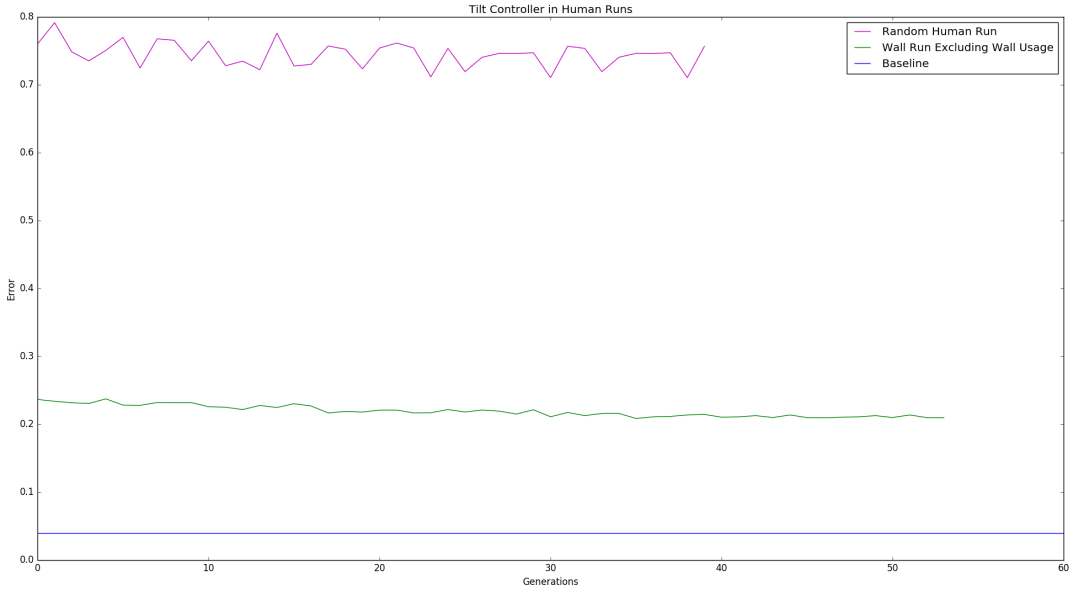

Figure 20: Tilt control parameters are not sufficiently able to approach a satisfying approximation for a human player in general.

6.6.2 Balance Control: Center of Mass Control PID-Gains

Like in the previous experiment, the balance control parameters also cannot influence the approximation towards a low error. During the AI experiments, the balance control could compensate for the tilt control parameters as seen in Figure 15. Here, both controllers do not affect the approximation much. However, in case of the wall run, an overall larger error can be observed in comparison to the error of the tilt control experiment in Figure 20.

The purple random human run converged after 38 generations with an error of 0.8054. The green wall run converged after 62 generations with an error of 0.47358. The wall run started with an error of 0.5170 and shows an improvement in error of 0.0434.

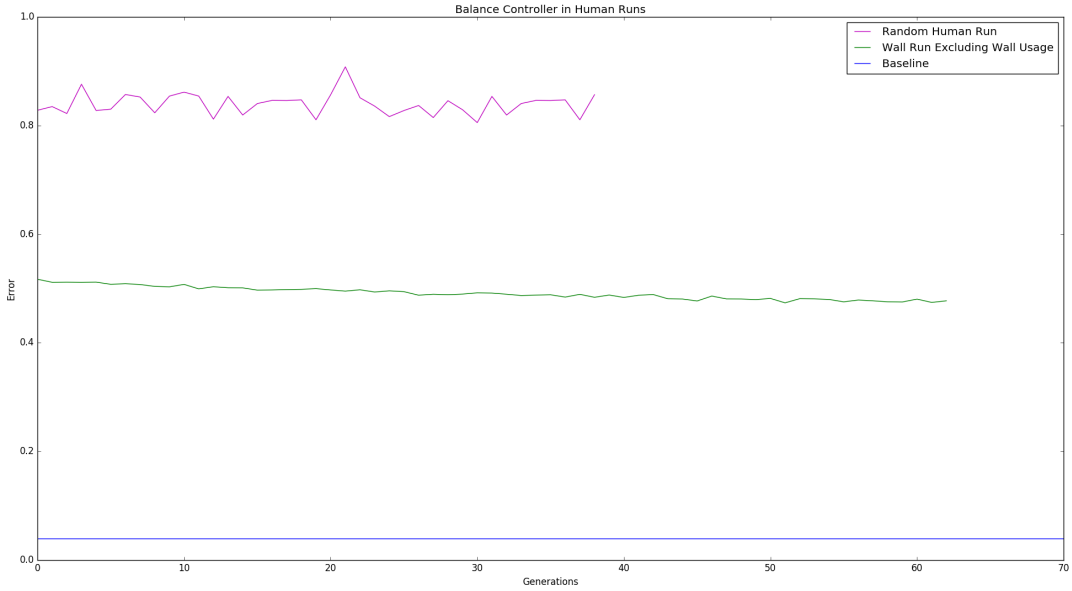

Figure 21: Balance control parameters are also not sufficiently able to approach a satisfying approximation for a human player in general. Also a lower impact on the overall approximation like in the AI experiment series can be seen.

6.6.3 Combining Tilt and Balance Control Parameters

When combining the tilt and balance control parameters, a better approximation can bee seen than in the previous two experiments, where only either the tilt or balance controller was optimized. For the wall run without wall usage, convergence was reached after 113 generations with a final error of 0.1389. The initial error was 0.2398, meaning that the error could be improved by 0.10008, but still has a distance of 0.1059 to baseline error.

The purple random human run started with an error of 0.7897 and converged with an error of 0.6893 after 135 iterations. This is an improvement of 0.1004, but still above the baseline by 0.6563. Also some spikes within the first 20 generations are seen, causing a maximum error of 0.9983 during the optimization process.

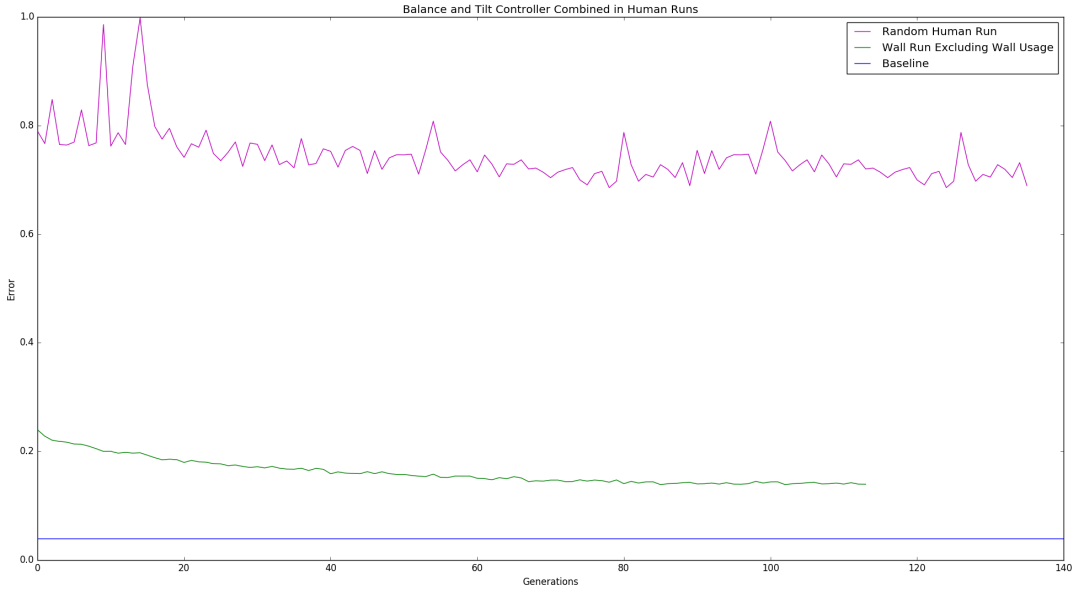

Figure 22: Tilt and balance control parameters in combination are also not sufficiently able to approach a satisfying approximation for a human player in general, but surpass the quality of the approximation of the single controller experiments. At the same time around twice as many iterations are needed for an approximation of similar quality.

6.6.4 Role of Wall Usage

As already seen in the AI-related wall experiment in Section 6.5.5, the wall factor $wallUsage$ plays a significant role when trying to approximate replays, in which the player made heavy use of the wall. Here is also the best approximation to a human replay possible. Comparing learning on a normal replay with learning on a replay, which is trying to stick to walls as much as possible is depicted by the green and purple lines. The red line shows a random human run. It is still not close to the baseline, but also optically seems to mimic the playing behavior well for the most part of the level.

The only relevant errors are at two corners in a level, where the AI can switch walls faster and more precise than the human player. Apart from that, an approximation after 110 generations, which is still above the convergence error baseline, but still a good approximation compared to the other human replays and at convergence with a final error of 0.05103 is the best approximation of a human player in all of the experiments. Learning the wall run without wall usage, indicated by the purple line, still shows an improvement over time, but was aborted after 83 generations with an error of at least 0.1017. The random human run indicated by the red line shows no particular improvement after 33 generations and generated an error of at least 0.1821.

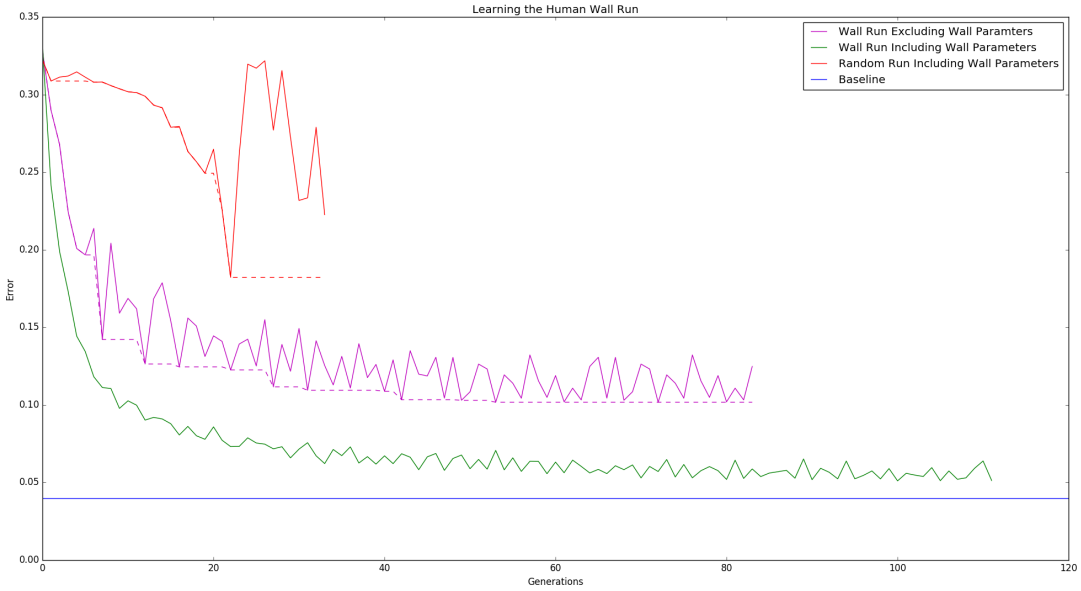

Figure 23: The difference between learning from a normal replay and learning from a replay, where the player is using a path as close to the walls. A replay with heavy wall usage can be approximated fairly well. The dashed lines indicate the current best solution for each generation.

6.6.5 Including Pathfinding Parameters

Additionally to the wall factor, another interesting observation is the change of the approximation when taking the other pathfinding parameters into account. The use of diagonals and outer diagonals does not have such a huge impact as the admission of approaching holes. However, similar to the influence of the $wallUsage$, seen in the prior section in Figure 23, the pathfinding parameters increase the whole approximation to a better quality overall, while the continuous parameters of the tilt and balance control do not influence the approximation that much.

In Figure 24 the red line indicates a run with only the PID parameters and not optimizing wall, hole or diagonal parameters. The purple line indicates a wall run without optimizing wall parameters, but using diagonal and hole parameters. The green line indicates a human wall run optimizing every parameter. Similar to the previous experiment in Section 6.6.4, an approximation of similar quality was reached, although not crossing the baseline error.

The red run had an error of at least 0.1142 after 115 generations. The purple run had an error of at least 0.0694 after 136 generations. The green run had an error of at least 0.0521 after 127 generations. The blue baseline is not visible due to scaling issues. States were averaged over 15 runs in this final experiment.

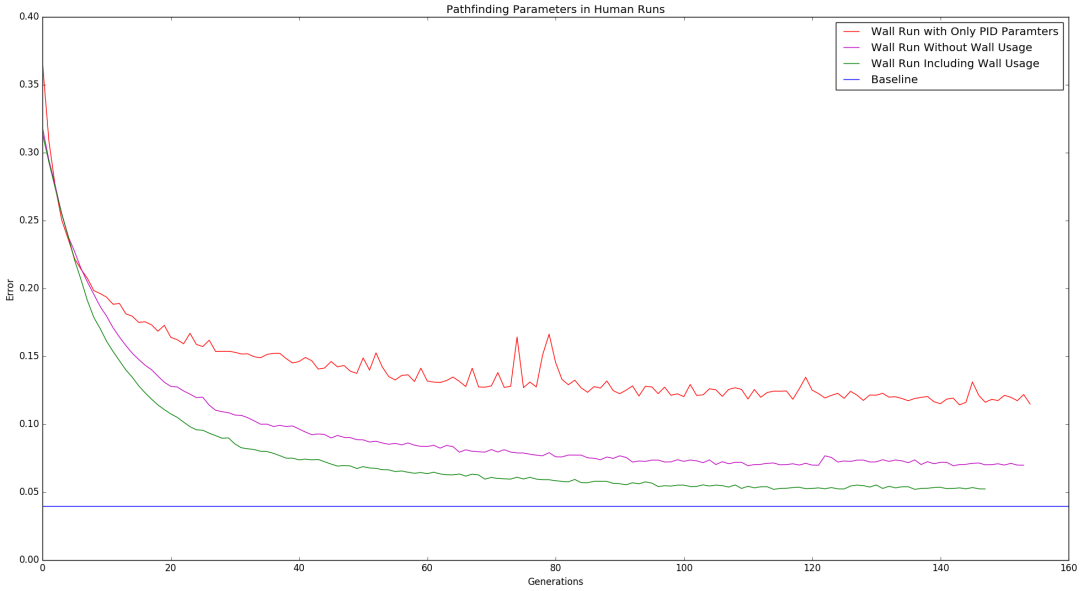

Figure 24: The use of diagonals, hole parameters ad wall usage have a major impact on the approximation.

7 Results and Conclusion

This chapter evaluates the results of the experiment in learning the AI's own behavior and the experiment of learning human player behavior. Afterwards a comparison is drawn between these results which leads to the final discussion of findings.

7.1 Results of the AI Learning its Own Behavior

The AI-related experiments, where the AI tried to optimize to its initial values, were all successful in general. Meaning that the learning process could improve the error at least as low as the baseline. When only optimizing the balance controller as in Figure 15 it can be seen, that this controller has not as much influence on the overall quality of the approximation as the tilt controller. While the tilt controller can compensate more for non-optimal balance parameters (see Figure 14), the balance control parameters can only reduce the error to a satisfying value when the tilt control parameters are already optimal, and hence does not show such high compensation potential.

A non-optimal balance controller can be met by the tilt controller to reach an error of 0.0787, while a non-optimal tilt controller can be met by the balance controller with an error of only 0.2231. A reason for this is the cascading design of tilt and balance controller, where the tilt controller first influences the error signal before passing it to the balance controller. Meaning that the balance controller by design expects a signal propagated by the tilt controller instead of a raw signal. that the tilt controller is more important for approximating sample runs, because it is able to cause a lower overall error with non-optimal balance control values.

7.1.1 Importance of Wall Usage and Pathfinding Parameters

Only when Boolean parameters were purposely changed, the error baseline could not be met. This means that the PID-controller cannot compensate for factors such as heavy wall usage. A reason for this is that the pathfinding algorithm plans a way without wall usage when this flag is set to $false$ and vice versa. Hence tuning parameters in order to to make use of walls, while the pathfinding algorithm neglects walls, is not possible. This also indicates, that the pathfinding parameters play a significant role while controlling the ball through the labyrinth.

The experiment in Section 6.5.5 showed, that it is impossible to let the tilt and balance controllers compensate for the usage of walls. The error could be reduced by 0.14332 when learning wall usage without the wall usage flag, but there was still a distance to the baseline of 0.10332. This indicates that a wall-favoring replay cannot be approximated from a wall-avoiding player and vice versa.

Similar behavior as in the wall factor experiment is shown when turning off the usage of diagonals in the experiment in Section 6.5.6. A run, which employs inner and outer diagonals, cannot be approximated by a player using only basic 90° turns.

7.1.2 General Difference in Number of Parameters

The higher the number of parameters to be optimized, the more generations for convergence are needed. With 3 parameters to be optimized, convergence was met after 24 respectively 28 generations while the error threshold was already crossed somewhere below 20 generations. Using the 6 controller parameters, convergence was met after 52 parameters. The optimization of all 9 continuous parameters needed 126 generations during the AI-related experiments.

7.2 Results of Learning Human Player Behavior

Learning on human runs was in general not as successful as learning from AI examples. The random sample runs from a sample group of 10 levels conducted at the Multimedia Communications Lab at Darmstadt University of Technology showed only limited capability for using it as a training set.

Therefore a human run was recorded, which was purposely close to an AI run to test if the AI setup is in principle able to improve when learning on human runs. It has been shown in the experiments in Section 6.6.4 and Section 6.6.5, that this human wall run could be used to reach a solution with respect to a certain error. Although the error baseline was never met, the learning process showed an improvement in these experiments.

Regarding the controller-related experiments in Section 6.6.1, 6.6.2 and 6.6.3 similar observations as in the AI experiments were made. Optimizing only the tilt controller leads to an overall smaller error than only optimizing the balance

controller. The error in the tilt experiment was at best 0.20849, while the error in the balance controller experiment was at best 0.47358.

7.2.1 Importance of Wall Usage and Pathfinding Parameters

In all human experiments, if an improvement of the error was seen, it was mainly caused by optimizing the non-controller-related parameters as the usage of walls, holes and diagonals. The PID-controller parameters did not cause a significant improvement.

In the wall-related experiment in Section 6.6.4 the error of the wall run could be improved to 0.05103, while in the controller-related experiment in Section 6.6.3 the best error was only 0.1389. When employing all parameters an error of 0.0521 could be reached. This is a sign that the quality of approximating human human runs mainly relies on the path parameters.

7.3 Comparison and Conclusion

In the AI-related experiment series it was shown, that the PID-controller parameters were as responsible as the non-controller parameters. It was also shown, that the Boolean parameters for using walls, holes and diagonals have to be employed when trying to learn runs, where these parameters were set to *true* before. Otherwise the approximation did not surpass a certain error.

For the human experiments the PID-controller variables did not seem to have a significant impact on the learning process. In the combined controller experiment for the AI in Section 6.5.3 the error could be improved by 0.15035, where in the human experiment for optimizing both controller in Section 6.6.3 the error could be improved only by 0.10008 in the best case. The final error in the human experiment was still 0.1389 while the AI experiment had a starting error of 0.17467. Meaning the final error of the best human tilt and balance controller runs where in a region where the AI-related runs were starting. This indicates that the two controllers are not as suitable for approximating human behavior than for approximating AI behavior.

Planning its way through the labyrinth, the artificial intelligence always follows the shortest path with respect to certain parameters such as wall factor, etc. At the same time the controller variables cannot really change on which path the ball will roll, but only how it will deviate from it. As a result, the corrected path will still be computed according to the current parameters and the AI plays the game using ideal paths and is in general strong goal-oriented playing behavior.

In contrast to this, a human player does not show such goal-oriented playing behavior and tends to maneuver through the labyrinth in a more arbitrary fashion. Learning this kind of behavior is not possible for the AI by design, since this style is incompatible with the path planning routine. The success in approximating the human wall run as a purposely target-oriented example supports this result.

In summary, for learning human behavior the controller seemed not to be able to increase the quality of a solution and only the controller-independent parameters are suitable to improve an approximation. In general the human player experiments needed more iterations, even when learning with low parameters. A direct convergence criterion was also never met. Rather, the optimization was aborted through a derived convergence criterion, causing more iterations.

8 Summary and Outlook

This chapter sums up the main concepts of this thesis and provides some ideas with respect to improvement of the results created in this work.

8.1 Summary

In this work we investigated if the controller based artificial intelligence in the exergame BalanceFit is able to approximate player behavior on the basis of replay data. Using a representation based on Markov decision processes and evolution strategy as an optimization technique, we ran an experiment series to determine relevant parameters and the quality of the approximation. The experiments in a first series were designed to approximate behavior of the artificial intelligent itself. The experiments in a second series were designed to approximate a human player.

Learning the behavior of the artificial intelligence is possible in general. Depending on the number of parameters and initial values, the optimization process differs in iterations, but still leads to an adequate result. When pathfinding or wall usage flags are taken into account, the approximation was only sufficiently possible when using these same flags during optimization.

On the other hand, learning the playing style of a human player is not always possible, or at least did not lead to a significant improvement within a range of iterations. Most replays could not be sufficiently approximated. However, some exceptional replays, which resemble the style of the artificial intelligence to some extent, like balancing next to walls, could be approximated sufficiently to show the learning potential of the AI. The controller-independent parameters as wall factor and the pathfinding parameters are a main reason for the overall success of approximating a human player. Therefore the tilt and balance controllers are in general not influential enough, such that a human player behavior can be approximated.

8.2 Outlook

For approximating human playing, it would be of interest if the current setup of the artificial intelligence in form of a tilt and balance PID-controller is able to learn the corresponding behavior of a human in general. Taking into account that the controller-independent parameters for pathfinding and wall usage have the main impact on the approximation, while the gain parameters both for tilt and balance control do not play decisive roles when optimizing for human behavior, a redesigned controller setup could be investigated.

Regarding the pathfinding routine the underlying shortest path algorithm of the AI works always goal-oriented and shows no arbitrary behavior. Human players however show more arbitrary behavior and often times deviate from an ideal path. Therefore simulating a less goal-oriented behavior and using non-ideal paths can result in a better approximation for a human style playing style.

Also changing parameters not only at the start but during a run could be a way of approaching this problem. In the extreme case, a parameter update at every time step could lead to a better approximation. BalanceFit as an already existing game was used for our experiments, and doesn't allow to use it as simulation environment to an extent, which would allow this kind of experimental setup. A simulation environment using a similar simulation of physics might allow a better investigation.

Given that the current artificial intelligence would be theoretically sufficient for approximating human player behavior, it would also be of interest if the number of optimization iteration can be decreased, either by changing the optimization method completely or by bootstrapping the optimization process.

9 Appendix

References

[1] K. J. Aaström and T. Hägglund. *PID Controllers: Theory, Design, and Tuning*. 1995.

[2] Jonathan Baxter, Andrew Tridgell, and Lex Weaver. *Learning to Play Chess Using Temporal Differences*. *Machine Learning*, 2000.

[3] Richard Bellman. *A Markovian Decision Process*. *Indiana Univ. Math. J.*, 1957.

[4] Richard Ernest Bellman. *Dynamic Programming*. 2003.

[5] Hans-Georg Beyer and Hans-Paul Schwefel. *Evolution Strategies A Comprehensive Introduction*. 2002.

[6] Felix Biemüller. *Entwicklung und prototypische Implementierung von Konzepten und Methoden zur Simulation menschlichen Verhaltens bei der Durchführung von spielerischen Übungen zum Training der Balance*. Bachelor Thesis, Darmstadt University of Technology, 2015.

[7] Christopher M. Bishop. *Pattern Recognition and Machine Learning*. 2009.

[8] Ingo Boersch, Jochen Heinsohn, and Rolf Socher. *Wissensverarbeitung - eine Einführung in die künstliche Intelligenz für Informatiker und Ingenieure*. 2007.

[9] Stephen Boyd and Lieven Vandenberghe. *Convex Optimization*. 2004.

[10] J. E. Bresenham. *Algorithm for Computer Control of a Digital Plotter*. *IBM Systems Journal*, 1965.

[11] Pua Y Clark RA, Bryant AL, Bennell K McCrory P, and Hunt M. *Validity and Reliability of the Nintendo Wii Balance Board for Assessment of Standing Balance*. *Gait Posture*, 2014.

[12] Thomas H. Cormen, Clifford Stein, Ronald L. Rivest, and Charles E. Leiserson. *Introduction to Algorithms*. 2001.

[13] Stefan Dillmann. *Entwicklung und Vergleich von Methoden zur Pfadsuche unter Berücksichtigung veränderlicher Randbedingungen*. Bachelor Thesis, Darmstadt University of Technology, 2014.

[14] Damien Djaouti, Julian Alvarez, Jean-Pierre Jessel, and Olivier Rampnoux. *Origins of Serious Games*. In *Handbook of Research on Improving Learning and Motivation through Educational Games: Multidisciplinary Approaches*. 2011.

[15] Aleksandra B. Djurisic, Aleksandar D. Rakic, E. Herbert Li, Marijan L. Majewski, Nenad Bundaleski, and Bozidar V. Stanic. *Continuous Optimization Using Elite Genetic Algorithms with Adaptive Mutations*. In *Selected Papers from the Second Asia-Pacific Conference on Simulated Evolution and Learning on Simulated Evolution and Learning*, 1999.

[16] Pedro Domingos. *A Few Useful Things to Know About Machine Learning*. *Commun. ACM*, 2012.

[17] A. E. Eiben and C. A. Schippers. *On Evolutionary Exploration and Exploitation*. *Fundam. Inf.*, 35(1-4):35–50, August 1998.

[18] Imran Ghory. *Reinforcement Learning in Board Games*. Technical report, Department of Computer Science, University of Bristol, 2004.

[19] Jan Giese. *Entwicklung von Konzepten und Methoden zur automatischen Generierung von personalisierbaren spielerischen Trainingseinheiten im Balancetraining*. Bachelor Thesis, Darmstadt University of Technology, 2013.

[20] Barbara M. Hayes and William Aspray. *Health Informatics. Patient-Centered Approach to Diabetes*. 2010.

[21] Mohammed Moussetad Houda Mouaheb, Ahmed Fahli and Said Eljamalic. *The Serious Game: What Educational Benefits? Procedia - Social and Behavioral Sciences*, 2012.

[22] Ronald A. Howard. *Dynamic Programming and Markov Processes*. 1960.

[23] Walter Isaacson. *The Innovators. How a Group of Hackers, Geniuses and Geeks Created the Digital Revolution*. 2014.

[24] Brijnesh J. Jain, Hartmut Pohlheim, and Joachim Wegener. *On Termination Criteria of Evolutionary Algorithms*. In *Proceedings of the 3rd Annual Conference on Genetic and Evolutionary Computation*, 2001.

[25] Lodewijk Kallenberg. *Finite State and Action MDPS*. 2002.

[26] Wolfgang Bösche Klaus Bredl. *Serious Games and Virtual Worlds in Education, Professional Development, and Healthcare*. 2013.

[27] Ray Kurzweil. *The Singularity Is Near: When Humans Transcend Biology*. 2006.

[28] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. *Playing Atari with Deep Reinforcement Learning*. In *NIPS Deep Learning Workshop*. 2013.

[29] Philipp Müller. *Entwicklung von Methoden und Konzepten zur Adaption digitaler Trainingssysteme basierend auf Algorithmen zur automatischen Generierung von Spielinhalten*. Bachelor Thesis, Darmstadt University of Technology, 2013.

[30] Ather Nawaz, Nina Skjæret, Kristine Ystmark, Jorunn L. Helbostad, Beatrix Vereijken, and Dag Svanæs. *Assessing Seniors' User Experience (UX) of Exergames for Balance Training*. In *Proceedings of the 8th Nordic Conference on Human-Computer Interaction: Fun, Fast, Foundational*, 2014.

[31] John Newcombe. *The Genetic Algorithm Framework for .NET*. [Online; accessed 28-January-2017].

[32] Peter Olofsson. *Probability, Statistics, and Stochastic Processes*. 2005.

[33] Dae-Sung Park and GyuChang Lee. *Validity and Reliability of Balance Assessment Software Using the Nintendo Wii Balance Board: Usability and Validation*. *Journal of NeuroEngineering and Rehabilitation*, 2014.

[34] Goncalo Pereiraa, Antonio Brissona, Rui Pradaa, Anaiva Pa, Francesco Bellotti, Milos Kravcikc, and Ralf Klamma. *Serious Games for Personal and Social Learning & Ethics: Status and Trends*. 2012.

[35] Stuart J. Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. 2003.

[36] Hans-Paul Schwefel. *Evolution Strategy and Numerical Optimization*. PhD thesis, 1974/1975.

[37] L.G. Silva, Arismar Cerqueira S., and Carlos H. da Silva Santos. *Development of Tri-Band RF Filters Using Evolutionary Strategy*. *AEUE - International Journal of Electronics and Communications*, 2014.

[38] David Silver, Aja Huang, Christopher J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. *Mastering the Game of Go with Deep Neural Networks and Tree Search*. 2016.

[39] Jeff Sinclair, Philip Hingston, and Martin Masek. *Considerations for the Design of Exergames*. In *Proceedings of the 5th International Conference on Computer Graphics and Interactive Techniques in Australia and Southeast Asia*, GRAPHITE '07, 2007.

[40] Herbert Spencer. *The Principles of Biology*. 1864.

[41] Richard S. Sutton and Andrew G. Barto. *Introduction to Reinforcement Learning*. 1998.

[42] Gerald Tesauro. *Temporal Difference Learning and TD-Gammon*. *Commun. ACM*, 1995.

[43] A. M. Turing. *Digital Computers Applied to Games*. In *Faster Than Thought*. Pitman Publishing, 1953.

[44] Christian Sebastian Loh, Yanyan Sheng and Dirk Ifenthaler. *Serious Games Analytics. Methodologies for Performance Measurement, Assessment, and Improvement*. 2015.