Universiteit
Leiden
The Netherlands

Computer Science

A Heuristic Approach For Goal-Oriented Agent Navigation In A Continuous Environment.

Pollux Taal

Supervisor:
Matthias Müller-Brockhausen

BACHELOR THESIS

Leiden Institute of Advanced Computer Science (LIACS)
www.liacs.leidenuniv.nl

15/01/2026

Abstract

The research is aimed at goal-oriented agent navigation in a continuous and physics-driven environment where traditional path-finding methods are difficult to use. Movement in this environment is influenced by indirect control, caused by momentum and gravity. The paper proposes a blend of three heuristic navigation strategies: one goal-oriented and the other two safety-oriented. The heuristics are combined using weights, creating a heuristic navigation strategy. This approach is developed in a simulation environment motivated by the video game "Super Monkey Ball", in which a player is tasked with tilting geometry so that a gachapon ball with a monkey in it rolls into the goal. The results show that heuristic blending can enable navigation in simple and moderately complex environments, while revealing that the performance is sensitive to weight configuration and stage geometry. For more complex stages, the proposed heuristic combinations are insufficient.

Contents

1 Introduction

Over the past two decades, Game Artificial Intelligence (Game AI) has undergone significant development, driven by the increase in computational power, better physics simulation and player expectation [YT18]. Modern games aim to simulate even more realistic interactions, which introduces new complexity. Amidst technological advancement and growing playing expectations, agent navigation remains a challenge.
Current trends around Game AI shift their focus to machine learning and deep learning [Chi24]. In this research, the aim is to investigate decision-making approaches that rely on explicit heuristics and reactive control, rather than learning-based methods.
To study the application of a heuristic approach to navigation, the research is done in a controlled but challenging setting emulating the environment of Super Monkey Ball. Super Monkey Ball is a physics-based platforming game where the player is represented by a monkey in a gachapon ball. In the main game-mode the player is tasked with rolling the ball into the goal by tilting the geometry of the level (also referred to as the stage). Because the game takes place in a continuous, physics-based environment, movement is not only continuous, but also momentum-based. The research allows for analysis of the agent's behavior and how the implemented navigational strategy works in a physics-based, continuous environment.
The control space is minimal, with the analogue stick as the sole parameter; yet the game poses a challenge for game-AI.
Agent movement is often tackled using traditional pathfinding techniques such as Dijkstra's Algorithm, A*, and Depth-first or Breadth-first search [GR03]. Most pathfinding algorithms work within a discrete and well-structured graph. In continuous environments, the likes of Super Monkey Ball force, momentum, and stage dynamics make it hard to apply traditional graph-based pathfinding techniques directly, creating a discontinuity between traditional methods and real-world applications. While classic algorithms compute exact and optimal solutions, heuristic methods aim to approximate good solutions more efficiently by using informed estimates of distance or cost.
This thesis proposes a heuristic goal-based navigation technique based on reactive control.
Heuristics can offer an alternative approach in this setting; instead of planning optimal routes, heuristic strategies can approximate movement. Heuristics aim to find a good solution to a problem in a reasonable amount of computational time. Heuristic algorithms give nearly the right answer or provide a solution not for all instances of the problem [Kok05].

1.1 Research Question

The main research question of this thesis is:
How can we use game-AI to create autonomous agent navigation for Super Monkey Ball
The research is aimed at the goal-based navigation of a physics-based continuous environment modeled after Super Monkey Ball. In this environment, momentum and gravity make traditional pathfinding complex; in this setting, heuristics can be applied to offer an alternative approach. For the research, a goal-based heuristic has been implemented as well as two safety heuristics. By assigning weights to the three heuristics, we hope to create a heuristic that is able to solve levels. This research will give insight on trade-offs between aggressive goal-oriented movement and safety-driven strategies, what blend of the aforementioned heuristics provides the best results, and finally,

the suitability of these specific heuristics in the specified environment.
A more specific sub-question is therefore:
How do different weight configurations influence success rate, distance metrics, and completion time in a heuristic approach to agent navigation in a continuous physics-based environment?

2 Related Work

Navigation can be separated into two different categories, heuristic approaches and classical approaches [AK23]. Conventional methods are often limited by several factors, such as high complexity and dependency on accurate environmental data [Kok05]. Factors that contribute to this are high-dimensional search spaces, dynamic and uncertain environments, trade-offs between optimality and computational efficiency, and scalability [VG25]. In a survey about heuristic and classical approaches to path planning in the field of robotics, we can see that the popularity of heuristic approaches to path planning in robotics has risen because of these limitations [AK23].
A definition of an agent is given as follows: Agents can be defined as autonomous, problem-solving computational entities capable of effective operation in dynamic and open environments [LMP03]. In-game AI, goal-based agents often control non-player characters (NPCs), which enables them to exhibit intellectual actions and behavior.
In many navigation systems, agents are designed to operate in a goal-oriented manner, where behavior is driven by an objective of reaching a specific target state. The following subsection (see section 2.1) explores common goal-oriented agent models and examines how they traditionally rely on explicit path planning techniques.

2.1 Goal-Oriented Agent Models

To increase the depth and appeal of computer games in general, the characters contained by the game should express some kind of intelligence, the characters should then be steered by agents that are motivated by goals and that are aware of how these goals can be reached [Haw00].
Goal-based AI agents are an approach in artificial intelligence where agents are programmed to achieve specific objectives [Gee24]. This approach works particularly well in complex environments where actions need to be adjusted dynamically as the agent can encounter unexpected obstacles and changes.
Goal-based AI agents work on a few key concepts; Goals, planning, execution, and adaptation [Gee24]. Goals are objectives that the agents aim to achieve; in the case of this paper, this denotes the completion of a stage. Planning refers to determining the sequence of actions required to achieve the aforementioned goal. Effective planning allows the agent to anticipate obstacles and avoid or conquer them. Execution indicates the agent's ability to carry out the specified planned actions, wherein the agent interacts with the environment and performs tasks that can bring it towards the goal. Adaptation causes the agent to change its plans as it interacts with the environment and gains new knowledge. Due to unexpected obstacles or changes, an adaptation in plans or strategy keeps the agent from straying away from its goal. A goal-based agent not only considers the consequences of its actions but also whether these actions are in line with the set goals [RM10].
This research will concern reactive agents, which operate on immediate perception and predefined rules, without long-term planning. Reactive, goal-driven agent models are particularly great in

dynamic environments, where the cost of maintaining detailed plans outweighs the potential benefits [WMMJ10].

2.2 Navigation in Continuous Environments

A continuous environment poses challenges not encountered in discrete environments. The continuous state space includes continuous variables, including position, velocity, acceleration, and direction, which cause the environment to have an infinite number of possible states. In terms of navigation, actions depend on previous choices as momentum and inertia play a role.
The problem of navigation in game AI is often tackled using grids that are composed of vertices or points connected by edges to represent a graph [ASK15]. Grids can be separated into two different categories: regular grids and irregular grids. A regular grid is a tesselation of regular polygons; the opposite are irregular grids. Examples of irregular grids are nav-meshes and way-points.
However, as Reynolds noted, such approaches effectively solve maze-like problems by creating a path [RA99]. In physics-based environments, paths alone do not address the issue of the task of movement. When continuous dynamics exhibit forces, the agent has to anticipate future consequences.
Without these predefined navigation structures, navigation relies purely on reactive planning mechanisms. Reactive algorithms operate on the current state of the agent and its perceived environment.
Using steering, reactive heuristic approaches allow the agent to respond dynamically to changes in its environment.

2.3 Heuristic Approaches in Agent Architecture

Real-time heuristic search methods can be developed to allow (multiple) agents to perform tasks within a large search space with limited computation [YT18]. The main drawback of navigation AI is that it only works in static environments, hindering the full potential of games [GS06]. This highlights a need for real-time navigation techniques in games.
At the base of a lot of real-time is an algorithm called Learning Real-Time A* (LRTA*) [BBSL11], building on A*. This algorithm requires a state graph, makes a planning and estimates costs.
The course taken in this thesis more closely follows a reactive approach to model player decisions. By evaluating multiple behavioral considerations using a heuristic and assigning weights, an action is chosen.

3 Methodology

This section will outline and substantiate the steps that have been taken in setting up the experiment and why they are necessary.

3.1 Research Design

In this study, multiple heuristic navigation techniques are implemented, tested, and compared by assigning weights to each. The environment in which the experiments are done is modeled after Super Monkey Ball. The goal of the study is to see how these heuristics and combinations of

heuristics perform in a continuous physics-based environment. The research is comprised of setting up the simulation (see section 3.2), implementing heuristics (see section 3.3), and finally running the simulation. After running the experiments, the relation between the weight and performance of each heuristic is studied. Performance is analyzed according to predefined success metrics.
The core research variables are the weights assigned to each heuristic, as the performance will be analyzed based on these weights. In future sections (see section 3.5), what the performance metrics collected and used are.
Super Monkey Ball is closed-source software; it cannot be used for our application, as control of the frame advancement and extraction of variables is needed in this research. The simulation is, therefore, not an exact clone of the systems implemented in the actual game. The simulation is made for this research as it gives a lot of flexibility and does not copy-right laws. The environment was built according to the needs of the research and provides a level of control and determinism that is not provided by any other Super Monkey Ball clones. The simulation has full control of the game loop, deterministic stepping, and allows for repeatable experimentation.

3.2 System Model

The simulation environment replicates essential features of Super Monkey Ball, and is built using Python, the Panda3D game engine [Tea22], the Bullet physics module [Pan24] in Panda3D, and Blender [Fou]. The stages were selected based on their difficulty, progressively getting more complicated.
In the following sections, the usage of these software tools will be explained and substantiated; a more technical and detailed look at the technical aspects of the software will be done in the "Simulation Setup" section (see section 3.4).
The Levels selected for experimentation are progressively more difficult by introducing new obstacles for the agent to overcome. In total, it consists of three stages, progressively getting more difficult. More explanation about the selected stages is given in a future section (see section 3.4.1).
The physics that play in Super Monkey Ball have been replicated by eye and feel, as Super Monkey Ball is a closed-source game, there was no access to the actual implementation. Therefore, the physics of the simulation might differ from the original game. Technical specifications of the physics are expanded upon in later sections (see section 3.4.2).

3.2.1 Environment Description

For the simulation, the Panda3D game engine was used for its flexibility, lightweight nature, and open-source. Because the physics engine in Panda3D is quite limited out of the gate, the open source Bullet Physics Engine was used. Panda3D has great support and integration with Bullet Physics for the included Bullet module, making it a great candidate.
Stage geometry was modeled in Blender and exported to the file format supported by Panda3D, EGG, using the YABEE exporter [TtYt]. In the simulation, the stage geometry is represented as multiple nodes using the " Bullet Physics "BulletRigidBodyNode" module.
Multiple simplifications are made from the real Super Monkey Ball. The tilting of the stage in Super Monkey Ball is assumed to be the actual stage geometry moving which lets gravity do the work. The simulation does not feature tilting geometry, such as the ones that appear in the actual game. Secondly, there is no score system, nor does the environment include bananas. And thirdly,

the testing environment does not include a camera that moves with the ball, which causes the tilt input to be more nuanced by rotating towards the direction of the balls velocity.
Technical details about the stage tilt mechanic are described and specified in later sections (see section 3.4.2).

3.2.2 Agent Architecture

The agent is represented as a rigid-body sphere and is created using the Bullet Physics "BulletSphereShape" module. The agent is controlled through the tilting of gravity; tilt controls are input through a 2D vector that acts like a control stick, closely replicating the system in Super Monkey Ball.
The agent receives the current control vector, the velocity, its position, the position of the goal, and a list that allows it to see whether or not there is ground nearby. These parameters are used by the heuristic in a weighted sum to select the best action, which the agent then executes.
The ball has a size, mass, and other parameters (see section 3.4.2 for a detailed overview) that cause it to approximate the behavior of the gachapon ball from Super Monkey Ball.
For the experimentation, the heuristics chooses an action every frame, with the simulation running at 30 frames per second. The action chosen consists of a 2-dimensional vector that represents the input of the control stick.

3.2.3 Goal Specification

The primary objective of the agent is to reach the goal within the given time limit of 60 seconds. To influence how that goal is reached, we have also optimized for other metrics.
The recorded outcomes that tell us how well the agent has performed in this case are the "Time" and "Completed" variables, where; The agent operates in a simulated environment that runs at 30 frames per second (fps) with discrete time steps:

$$c \in \{0, 1\} \textit{ is the Completed value.}$$

$$t \in \{0, 1, ..., 1800\} \textit{ denotes the time in } frames.$$

When the goal has been reached, the simulation for that specific level with the specified weights stops immediately. This is represented as a completed value of 1 in the data.

$$t \in \{1, 2, \ldots, T_{max}\} \textit{ with } T_{max} = 1800$$

$$c = 1$$

Other termination conditions are time-outs and fall-outs.
Each agent run ends after 60 seconds (1800 ticks/steps), this is called a time-out. In the data, this will be represented as a time of 1801 frames and a completed value of 0. We can define this as such:

$$t = 1801$$

$$c = 0$$

A fall-out occurs when the ball has fallen off the stage and reached a depth of -20 in-game units. In the data, a fall-out can be detected by a time less than 1801 frames, and a completed value of 0, defined as;

$$t \leq 1800$$

$$c = 0$$

Therefore, each simulation has three different outcomes. For each simulation, it can be evaluated with two variables: Time, represented as t, and Completed, represented by c. Formally, we can define it as;

$$(t,\, c) = \begin{cases} (t,\, 1), & \text{if the goal is reached at time } t \leq T_{max}, \\ (T_{max} + 1,\, 0), & \text{if a time-out occurs,} \\ (t,\, 0), & \text{if a fall-out occurs at time } t \leq T_{max}. \end{cases}$$

3.3 Heuristic Approaches Evaluated

For the experiment, three different heuristic approaches have been implemented for the agent navigation that are blended using weights. In the following subsections, they will be outlined, as well as giving further elaboration on why they were chosen.

3.3.1 Heuristic 1: Goal Oriented Movement

The purpose of the first heuristic is to simply move the ball to the goal. To achieve this a vector is calculated that can be used as a control vector to move the ball straight to the goal. This approach ignores any risk, and thus doesn't fare well on its own when obstacles or edges are involved.

$$\text{moveTowardsGoalHeuristic}(\mathbf{b}, \mathbf{g}) = \begin{cases} \dfrac{(x', y')}{\sqrt{x'^2 + y'^2}}, & \text{if } |x'| + |y'| > 1 \\ (x', y'), & \text{otherwise} \end{cases}$$

$$\text{where} \quad \begin{aligned} x &= g_x - b_x \\ y &= g_y - b_y \\ x' &= \max(-1, \min(1, v)) \\ y' &= \max(-1, \min(1, v)) \\ g &= \text{The coordinates of the goal} \\ b &= \text{The coordinated of the agent} \end{aligned}$$

3.3.2 Heuristic 2: Moving to safe ground from the control-vector perspective

The second heuristic approach takes the current control vector and evaluates whether or not the ball is rolling in the right direction by checking for ground around the ball (For mathematical definition see section 3.3.3). This heuristic returns a vector that points to the middle of the most ground it could find. This approach is aimed at keeping the ball on the stage, disregarding the need for rolling towards the goal. This heuristic helps by counteracting the aggressive approach of the first heuristic.

3.3.3 Heuristic 3: Moving to safe ground from the velocity perspective

The third heuristic approach works similarly to the second, except this time taking into account the velocity of the ball instead of the control vector. The velocity of the ball, unlike the control vector, cannot change immediately. This heuristic is implemented to reduce the risk of the ball rolling off the stage due to its momentum.
Let:

$$c = \begin{pmatrix} x \\ y \end{pmatrix} \quad \text{The control vector}$$

$$G = (g_0, g_1, \ldots, g_{23}), \quad g_i \in \{0, 1\} \quad \text{ground list}$$

$$\tilde{G} = GG \quad \text{circular extension of the ground list}$$

$$L^* = \arg \max_{\substack{L \subseteq \{0,1,\ldots,47\} \\ L \text{ continuous},\ \tilde{G}_i = 1\ \forall i \in L}} |I| \quad \text{Longest list of uninterrupted ground}$$

$$\text{if } |L^*| = 0 \quad \Rightarrow \vec{r} = (0, 1)$$

$$m = L^* \left(\left\lfloor \frac{|L^*|}{2} \right\rfloor \right) \quad \text{The middle element of the list}$$

$$\theta = \frac{360^\circ}{24} \times (m \bmod 24) \quad \text{Angle corresponding to midpoint}$$

$$c' = \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix} c \quad \text{The rotated control vector}$$

$$\text{preferMostGround}(\mathbf{c}, G) = normalize(c')^{1} \quad \text{The resulting vector}$$

3.3.4 The Blending of Heuristics

Using weights, the three control vectors are blended. The total of the weights cannot exceed 1.00. In the following formula the weights are represented as **w** followed by a number denoting to which heuristics they belong, Heuristics are denoted as **H** followed by a number that shows the type and a **x** or **y** that denotes the axis of the control vector. Finally $\vec{c}$ is the resulting control vector.

$$x = w_1 H_{1,x} + w_2 H_{2,x} + w_3 H_{3,x}, \tag{1}$$

$$y = w_1 H_{1,y} + w_2 H_{2,y} + w_3 H_{3,y}. \tag{2}$$

$$\vec{c} = \begin{pmatrix} x \\ y \end{pmatrix} \tag{3}$$

The blending of heuristics using weights is aimed at finding a balance between aggressive goal-seeking behavior and safe behavior that supposedly keeps the agent on the stage geometry. By analyzing the performance data, trends can be identified, relating the weight of the heuristic to the overall performance. Furthermore, the method is simple to implement and test, as well as exhibiting deterministic behavior.

[1] See section for 3.4.2 normalize function

As described, the influence of heuristics is determined by the assigned weights; the weights are increased in increments of 0.01. These weights have a value w where

$$w \in \{0.00, 0.01, 0.02, \ldots, 1.00\}.$$

In total, there are three weights, one for every heuristic; the total of the three weights (w_1, w_2, and w_3) cannot exceed 1.00. We can define this as:

$$w_1, w_2, w_3 \in \{0.00, 0.01, 0.02, \ldots, 1.00\},\ w_1 + w_2 + w_3 = 1.0$$

3.3.5 Baseline Methods

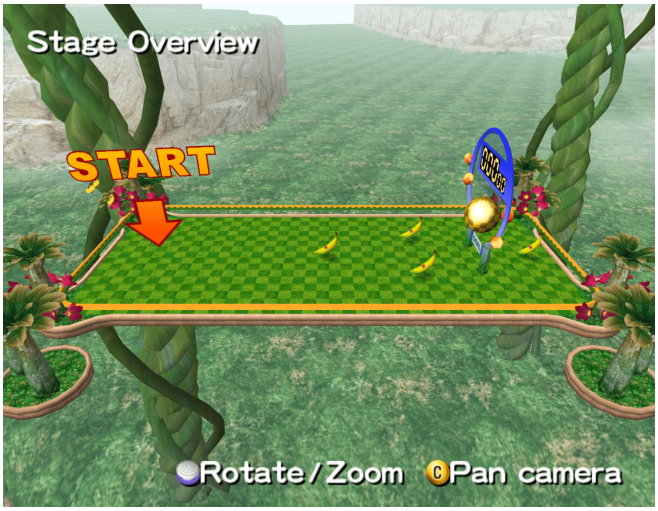
To evaluate the effectiveness of the proposed heuristic with its assigned weights, two baseline control methods are implemented. The baseline methods aim to place the effectiveness of the heuristic on the spectrum between naive and more engineered control.
The first baseline control method is a heuristic that produces a control vector that always sends the ball in a straight line forward. This heuristic does not incorporate any information and produces a vector that corresponds to a fixed control vector of $(0, 1)$. This baseline serves as a lower bound on the performance.
The second baseline control method is more nuanced and more closely represents the tested heuristic. It is composed of a goal-seeking function and a function that calculates the safest direction for the control vector. If forward is not a safe direction, the goal-seeking heuristic will have less impact on the control vector. This heuristic is built on the same heuristics (see section 3.3.1 and 3.3.2 used in the heuristic tested with weights.
The two baseline methods are chosen to represent contrasting levels of heuristics. The straightforward heuristic represents uninformed control, while the engineered heuristic demonstrates informed control with more nuance.

3.4 Simulation Setup

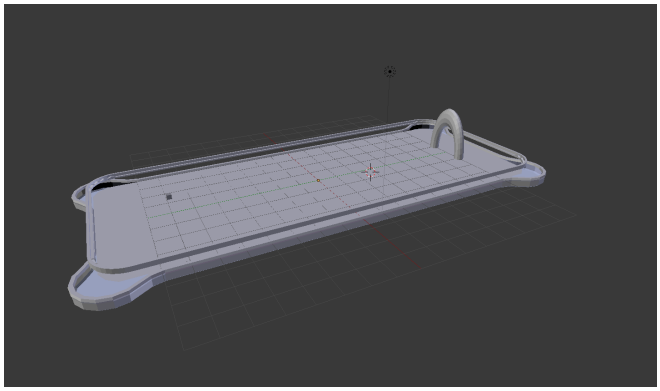
In total, three levels have been selected (see section 3.4.1), increasingly getting more difficult. The levels have been modeled in Blender 2.70 [Fou] and are exported to Panda3D through the supported EGG files using the YABEE exporting tool [TtYt]. For every combination of weights, all levels will be simulated, regardless of failure on the previous level. This means that the total number of trials per weight configuration is 3.
The simulation relies on two core technologies, Panda3D [Tea22] and the Panda3D Bullet Physics module [Pan24].
Panda3D is a scene-graph-based engine and includes a task manager for updating loops. In the experiment game-loop, this task manager is advanced in steps after calculating the control vector. Panda3D is also used for loading models.
For the rigid body dynamics, collision shapes, and physics simulation, the Panda3D Bullet Physics module is used. Bullet Physics is also used for ray-casting, which is used for creating a list of safe ground for the heuristic.
While the original Super Monkey Ball runs at a frame rate of 60, this simulation runs at 30 frames per second; this choice has been made in favor of faster simulation.

3.4.1 Level Selection

The levels chosen for the experiment are modeled after stages 1, 2, and 6 of the 'beginner' stages in Super Monkey Ball 1. In the results (see section 5), stage 1 is referred to as level 0, stage 2 is referred to as level 1, and stage 6 is referred to as level 2.
The stages were selected to increase in difficulty and introduce new obstacles. Later stages pose a significantly harder challenge for the agent.
Stage 1 (See figure 1) is the starting stage of the beginner levels. Used for getting the player acquainted with the controls, going straight will solve this level. The agent cannot fall off the stage geometry thanks to the guardrails; the only method of termination (see section 3.2.3) is a time-out. This level serves as an absolute baseline for what the heuristic must achieve: reach the goal in the given amount of time.



(a) As seen in Super Monkey Ball [Com22]



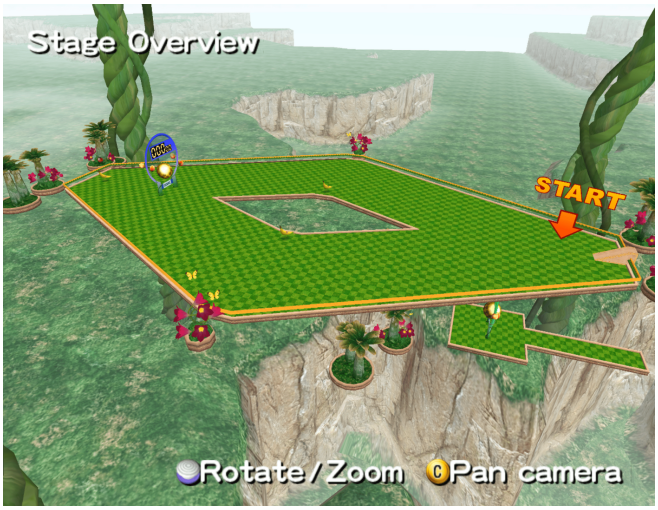
(b) Modeled for simulation using Blender

Figure 1: Stage 1, Plains

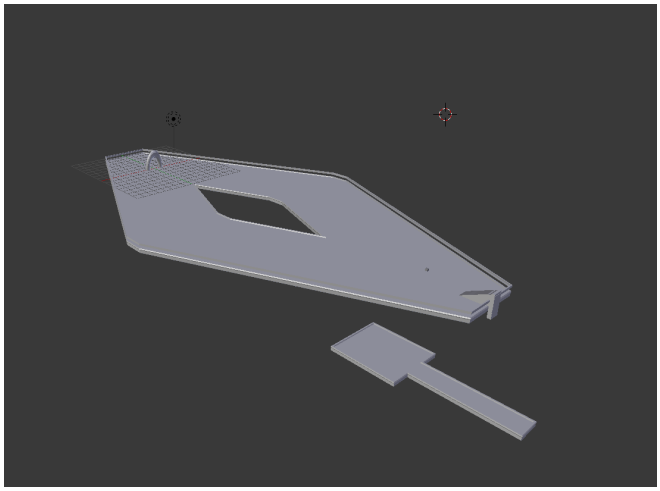
Stage 2 (See figure 2) introduces an obstacle in the form of a hole in the middle of the stage geometry. To reach the goal, the agent must avoid the gap by going around it. The secret goal found under the stage that advances to level 5 in Super Monkey Ball has been omitted.
Stage 6 (See figure 3) removes outer guard-rails and introduces verticality as well as the need to move away from the goal. As the agent does not account for the need to move away from the goal, this will pose a challenge. Verticality will introduce external forces on the agent, adding to the difficulty.

3.4.2 Simulation Of Physics

The simulation game simulated a tilting board by rotating the gravity vector rather than tilting the stage geometry.
Super Monkey Ball uses a custom physics system; all implemented physics are based on the visuals of the game. The gravity value has been set to 9.81, this is interpreted by Bullet Physics as $units/second^2$. The simulation is based on the principle $1\,Panda3D\,Unit = 1\,Meter$, emulating the gravity on Earth.

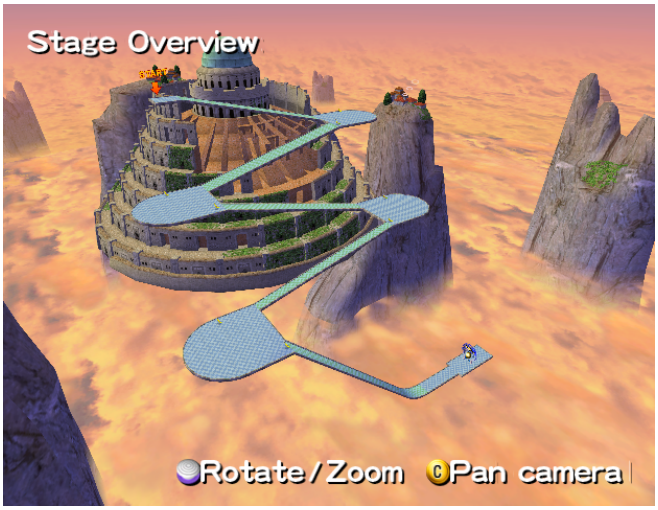


(a) As seen in Super Monkey Ball [Com22]

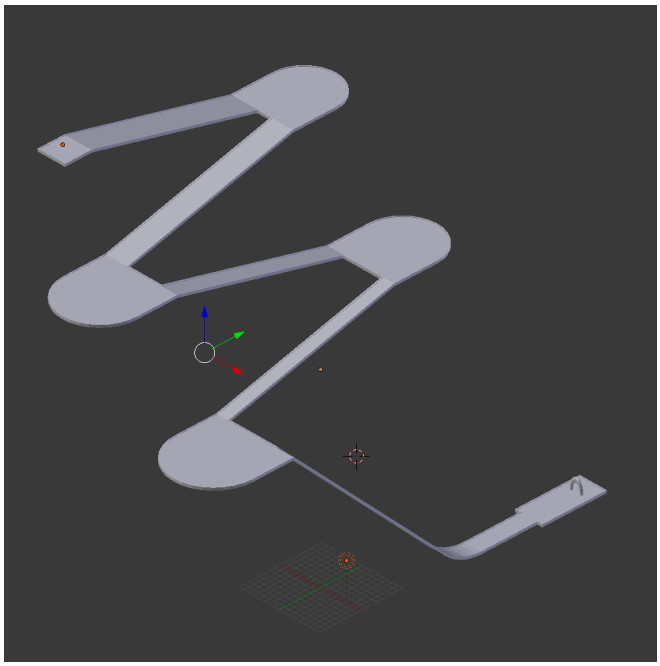


(b) Modeled for simulation using Blender

Figure 2: Stage 2, Diamond



(a) As seen in Super Monkey Ball [Com22]



(b) Modeled for simulation using Blender

Figure 3: Stage 6, Slopes

For the world, "BulletWorld()" is used, holding all physics bodies, managing gravity, and handling ray tests and collision queries.
The ball uses the "BulletSphereShape" for its shape and "BulletRigidBodyNode" for the body. The

following physics constants are applied:

$$\begin{aligned} \text{radius} &= 0.4 \\ \text{mass} &= 15.0 \\ \text{friction} &= 1.0 \\ \text{linear damping (rolling resistance)} &= 0.02 \\ \text{angular damping (spin decay)} &= 0.3 \\ \text{restitution (bounciness of the ball)} &= 0.7 \end{aligned}$$

The stage geometry consists of bullet collision bodies. The "fromCollisionSolids()" was used to convert the stage models' collision solids into Bullet rigid bodies. The following physics parameters are applied to the stage geometry:

$$\begin{aligned} \text{mass} &= 0.0 \\ \text{restitution (bounciness)} &= 0.5 \end{aligned}$$

The velocity of the ball is calculated by Bullet and is determined solely by external forces. Based on the input vector, the direction of gravity changes.
Let:

$$\begin{aligned} g &= 9.81 = \text{gravity magnitude} \\ \vec{c} &= \begin{pmatrix} \theta_x \\ \theta_y \end{pmatrix} = \text{the input vector} \\ \theta_x &\in [-1, 1] = \text{tilt input of the } x\text{-axis} \\ \theta_y &\in [-1, 1] = \text{tilt input of the } y\text{-axis} \\ k &= 0.5 = \text{tilt multiplier} \\ \vec{g} &= (g_x, g_y, g_z) = \text{resulting gravity vector} \end{aligned}$$

The input-vector $\vec{c}$ is first normalized using the following formula:

$$\text{if } \sqrt{\theta_x^2 + \theta_y^2} > 1, \quad (\theta_x, \theta_y) \leftarrow \frac{(\theta_x, \theta_y)}{\sqrt{\theta_x^2 + \theta_y^2}}$$

Secondly, the horizontal components of the gravity vector are computed;

$$g_x = kg\theta_x$$

$$g_y = kg\theta_y$$

The vertical component of the gravity vector is calculated using the following formula;

$$g_z = -\sqrt{g^2 - \left(g_x^2 + g_y^2\right)}$$

The resulting gravity vector is then;

$$\vec{g} = (g_x, g_y, g_z)$$

Finally, to prevent jitter, gravitational smoothing is applied, with the gravity value of the previous frame denoted as $oldG$ and the new gravity vector denoted as $newG$:

$$\vec{newG} = \vec{oldG} \times 0.8 + \vec{g} \times 0.2$$

3.5 Success Metrics

In this section, the different success metrics will be outlined and explained. There are five success metrics for measuring how close the ball got to the goal, the total distance covered, how far the ball was when it fell off or timed out, how much time it spent in a level, and finally, whether the heuristics succeeded.

3.5.1 Distance from the goal

Distance is measured in three ways: the closest the ball has ever been to the goal for that specific level and the distance when the ball has fallen off the stage or timed out. The first distance metric, the closest the ball has been to the goal, is the shortest distance the agent has reached to the goal, and gives insight into the effectiveness of the weight combination. The second distance metric shows how far from the goal the ball was upon losing the level. This can be useful to see whether the heuristics did a good job of keeping the ball near the goal. The third metric is the total distance covered by the agent; this will give insight into how efficient the path taken by the agent is. If the level has been completed, both of these metrics are 0.

3.5.2 Success Rate

The success rate is either 0 or 1. If a level has been completed, the success rate is 1; if it fails to do so, the result is 0.

3.5.3 Time

Finally, the time is measured in frames; the agent gets 60 seconds to complete the level, which is equal to 1800 frames in the simulation. A time of 1801 frames means that it has timed out. Time can be a double-edged sword; on one hand, the goal should be reached as fast as possible, but on the other, a longer survival time might be better than instantly rolling off the edge.

3.6 Grid Search

Testing of the weights is done using grid search, which is a method commonly used for hyperparameter optimization [LL19]. This method simply makes a complete search over a given subset or search space that satisfies a set of predefined boundaries (see section 3.3.4).
Grid search is an exhaustive and straightforward method that is easily reproducible. Due to grid search systematically evaluating all combinations of the parameters, it ensures that the optimal weight combination in the given bounds is identified.

3.7 Data Collection Procedures

The distance the ball has at the moment of termination (see section 3.2.3) of a run is extracted straight from the simulation from in-game variables, obtained by subtracting the goal position from the ball position.
Similarly, every frame, the distance between the goal and the ball are evaluated, and the shortest is stored. This denotes the closest the ball has been to the goal in a simulation.

The number of frames a simulation is running is stored in a variable in the main game loop. When this variable exceeds the max of 1800 frames, the simulation stops immediately.
When a simulation stops, all variables are re-initialized.

3.8 Statistical Analysis

To evaluate the collected data and identify meaningful patterns, a statistical analysis is conducted using a combination of descriptive summaries and visualization techniques.
To achieve the visualization, the Python package seaborn [Was24] is used.
Multiple forms of data representation are employed to best understand the results. It is done using table summaries, line plots, and heatmaps. Through these methods, we can make an interpretation of performance across weights; the combination of line-plots and heatmaps aims to highlight trends in the data. Tables aim to provide a guide for the data in the dataset and include the mean, median, and absolute best value in the dataset.
Visualization of the graphs and heatmaps is done using the Python Seaborn [Was24] library. No outlier treatment or data smoothing is applied during the analysis.

3.8.1 Descriptive Statistics

Descriptive statistics are the primary means of performance evaluation.
Tables (see table 1, 2, and 3) are constructed with three aspects of the dataset, mean, median, and best score. The agent did not complete level 2 for any weight combination; level 2 is evaluated in a different way from levels 0 and 1.
Line-plots and heatmaps (Visible in section 5) work together to show trends in data.
The line plots are created using the Pandas library [Teab] for data handling, and the Seaborn and matplotlib [Teaa] libraries for visualization. Data is separated by level. The x-axis represents the weight, and the y-axis represents the success metric. For each combination of weight and success metric, a new plot is produced. Seaborn's line-plotting function is used to compute and display the mean value for each combination, and on each line, a 95% confidence interval is shown.
In addition, heatmaps are used to study the combined influence of weights. A three-dimensional scatterplot is constructed where the three axes correspond to the weights and the color corresponds to the success metric, as shown in the legend. Furthermore, data is separated by level to ensure comparison within the same experimental conditions. No pre-processing or outlier treatment is applied to maintain the integrity of the data.

4 Experiments

To optimize our heuristic, we apply a grid search with a set of weights (see section 3.3.4). The experiment is designed to show how different weight combinations affect agent performance. All experiments are conducted with the same framework to ensure that there is consistency. The only variation between runs is the distribution of weights.
Per weight combination, one run is done per level, as the simulation is deterministic. For each weight, depending on level, the agent has the same starting position and attempts to reach the goal in the predefined maximum time limit of 60 seconds.

During the run, the agent evaluated its heuristics every frame and produced the control vector according to the assigned weights The simulation terminated when certain conditions were met (see sections 3.2.3).
For each run, data is recorded (see section 3.5) and stored, including weights and level in a csv file. The primary variable in this research is the weight assigned at every run. A grid search (see section 3.6) method is used to generate all possible combinations of weights with the given bounds.
Weight combinations are evaluated separately per level. The levels are selected on the criteria of progressively increasing in difficulty (see section 3.4.1), with more complex levels introducing new challenges.
In total, 15453 simulations have been run and recorded, meaning 5151 different weight combinations have been run on 3 different levels.

5 Results

Results in the tables (see table 1, 2, and 3) are sorted by level and the completion boolean. If a level has been completed, we want to optimize different parameters compared to uncompleted levels.
In completed levels (table 1 and 2), the heuristic performed best if the time and total distance are minimized, as this means a faster and more efficient finish. In incomplete levels (table 3), the importance of time and total distance metrics is up for debate, metrics that tell us the performance is the closest distance the agent has gotten to the goal and the distance from the goal at simulation termination.
The line plots and heatmaps are not separated by completion value and instead intend to show trends corresponding to weights. The axes of the graphs correspond to the weights.
Weight 1 represents goal-oriented movement (see section 3.3.1).
Weight 2 represents the safety heuristic from the perspective of the current control vector (see section 3.3.2).
Weight 3 represents the safety heuristic that reasons from the perspective of the agent's velocity (see section 3.3.3).

Table 1: Level 0, Completed

Success Metric	Mean	Median	Best Score
Time	190.50	186	185
Total Distance	14.27	14	14

Table 2: Level 1, Completed

Success Metric	Mean	Median	Best Score
Time	1137.52	1068	436
Total Distance	144.15	134	80

Table 3: Level 2, Not Completed

Success Metric	Mean	Median	Best Score
Time	912.79	804	N/A
Closest	38.28	36	1
Total Distance	228.16	242	N/A
Distance at Termination	50.55	50	22

5.1 Total Distance Covered

Heatmap analysis (see figure 4):
Level 0: When weight 1 is set to 0, the agent exhibits minimal movement towards the goal. The observed higher value in this case can be attributed to physics effects that result in small movements that add up over time. For a higher value than 0 for weight 1, the total distance covered converges to around 14 units, corresponding to the distance between start-position and the goal. This means that, for level 0, a small amount of weight 1 is enough to guide the agent to the goal.
Level 1: In this environment, a lower value of weight 1 is associated with an increase in total distance covered. This suggests a less risky or more conservative approach, leading the agent to take longer but safer trajectories. Low total distance covered in this level is often associated with failure (see the heatmap for level 1 in figure 12).
Level 2: A lower value of weight 1 is associated with a reduced total distance, enhanced further with the addition of a high weight 3. This effect is likely due to the agent not moving much because of the surrounding hazards. A high total distance with a relatively high weight 1 in this instance is caused by the verticality of the level.

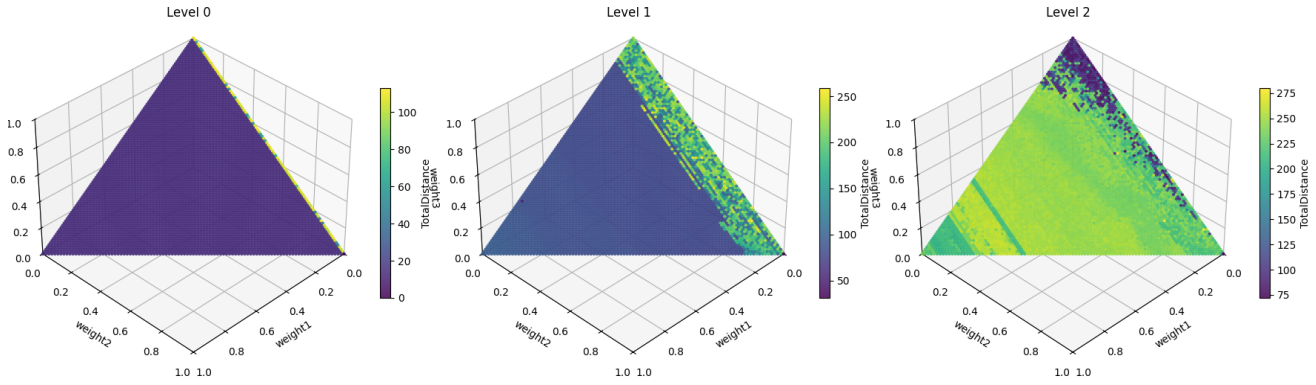


Figure 4: Heatmap of total distance covered

Trend analysis (see figure 5):
Weight 1 (see figure 5a) has a positive correlation with total distance in level 2, a negative correlation with level 1, and minimal influence in level 0, except at a very low value.
Weight 2 (see figure 5b) has a positive effect on distance in both levels 0 and 1 until very high levels, where it starts fluctuating. There is a weak and inconclusive impact on level 2.
Weight 3 (see figure 5c) Exhibits a positive impact on total distance for levels 0 and 1 and a

negative impact on level 2. The latter indicates that this safety heuristic dampens movement in dangerous areas.

(a) Weight 1 vs Total Distance (b) Weight 2 vs Total Distance (c) Weight 3 vs Total Distance

Figure 5: Effect of weight parameters on total distance covered across levels

5.2 Distance at Time of Simulation Termination

For this success metric, a lower value is generally better. A value of zero corresponds to the goal being reached while higher values indicate the agent was far away from the goal at the time of termination (see section 3.2.3 for termination criteria).

Heatmap analysis (see figure 6):

Level 0: Across most weight configurations, the goal is reached, thus the distance is zero. However, for a weight 1 of zero, the goal is often not reached, due to the agent not exhibiting any goal-seeking behavior.

Level 1: The best performance is observed here for a high weight 2 (around 0.9) in combination with a low weight 1 (around 0.1). Increasing weight 1 beyond this leads to worse outcomes, until the distance lowers again. This can be attributed to the agent falling through the gap in the stage (see section 3.4.1) at a higher speed, thus reaching closer to the goal under the stage geometry. At low weight 1 in combination with weights other than a weight 2 of 0.9, the agent seems to exhibit inconsistent or unstable behavior and have highly variable results.

Level 2: Due to the strong verticality in the level, performance improves with very high values of weight 1. A high weight 3 seems to result in low agent movement, causing a high distance at the time of termination. Because the heuristic tied to weight 2 works with the control vector, movement is still happening.

Trend analysis (see figure 7):

Weight 1 (see figure 7a). At level 0, the goal is almost always reached unless the weight is 0. For level 1, the performance fluctuates for a value under 0.2. Upon a high weight, the performance seems to increase; this is likely due to the ball falling off the stage at a higher velocity, thus getting closer to the goal, albeit under the stage geometry. A high weight value causes the agent to get closer to the goal in level 2, once again this is due to the verticality of the stage, with optimal results at $weight1 \approx 0.9$

Weight 2 (see figure 7b) has a relatively subtle effect, across all levels, although extremely high values have a negative impact on movement entirely. Overall, a slightly positive impact above a weight value of 0.6 can be observed in levels 1 and 2.

Weight 3 (see figure 7c) shows a highly negative impact on performance on level 2 and a slightly positive impact on level 1 past the midway point. Very high levels of weight 3 negatively impact performance because of the lack of movement.

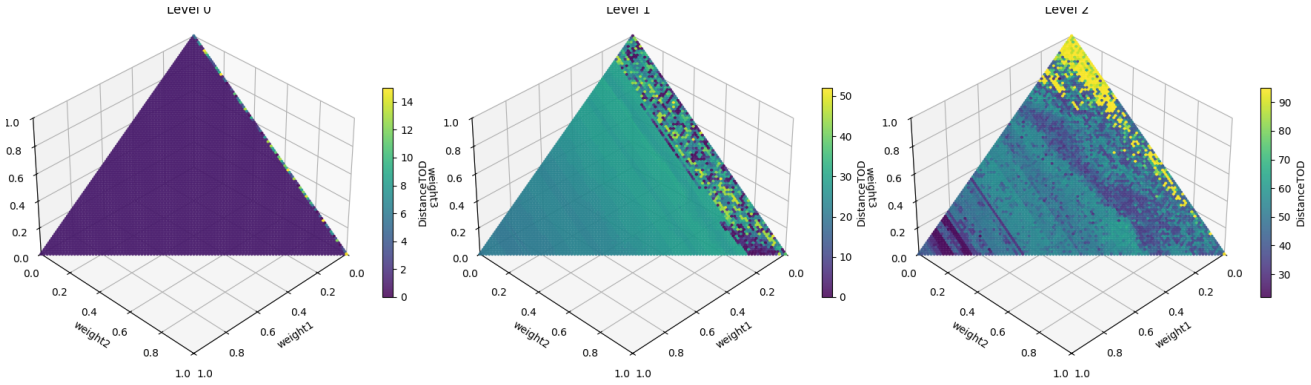
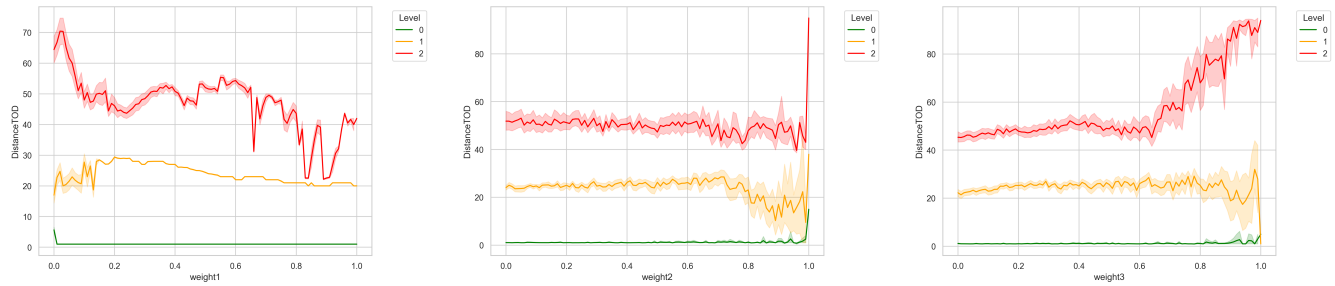


Figure 6: Heatmap of distance between agent and goal at the time of termination of the simulation



(a) Weight 1 vs Distance (b) Weight 2 vs Distance (c) Weight 3 vs Distance

Figure 7: Effect of weight parameters on distance at time of simulation termination across levels

5.3 Closest Distance the Agent Reached the Goal

The metric studied is the closest distance to the goal achieved by the agent during a simulation. A lower value indicates better performance, as the agent has gotten closer to the goal.

Heatmap Analysis (see figure 8:

Level 0: For nearly all weight combinations, including a very low weight 1 value, the agent has gotten close to the goal. A very large portion of the simulations have successfully finished.

Level 1: The best performance is observed when weight 1 is below 0.2. Variations in weights 2 and 3 have little effect on the outcome.

Level 2: Optimal results are observed when weight 1 is a high value, which is consistent with the verticality of the level, as mentioned previously. The worst results are observed when weights 1 and 2 are low (approximately under 0.2), and weight 3 is high; this combination leads to minimal agent movement.

Trend Analysis (See figure 9):

Weight 1 (See figure 9a), when very low, produced the worst outcome for level 0. In level 1, the

best performance is achieved with a low value of weight 1, in particular under 0.1. The performance in level 2 improves by increasing weight 1, with peak performance around a weight of 0.85 and 0.9.
Weight 2 (See figure 9b) has a slight positive impact in levels 1 and 2, indicated by a weak downwards trend in the graph. However, an extremely high weight has a negative influence on performance across all levels, likely due to limited movement.
Weight 3 (See figure 9c) has a positive impact on performance in level 1, and a negative influence on performance in level 2. This indicates that, when faced with a more hazardous level, the heuristic causes limited agent movement.

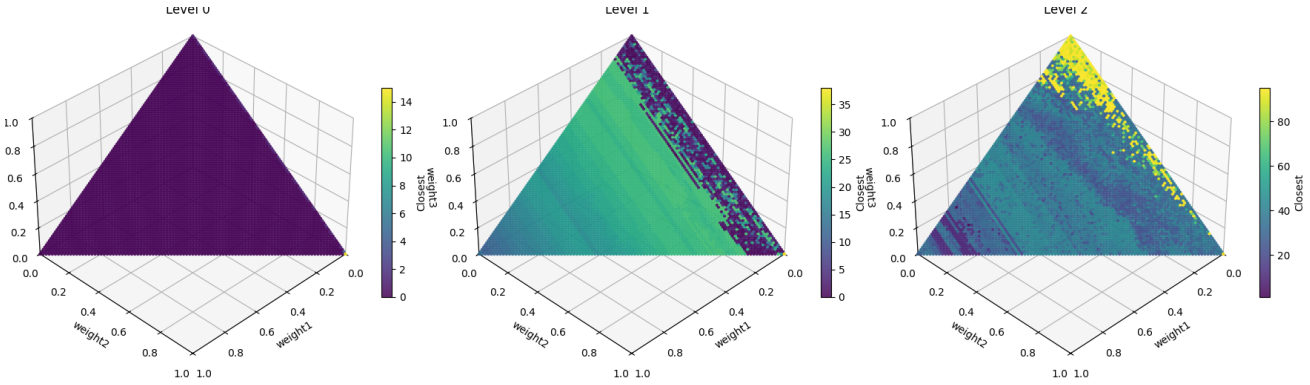
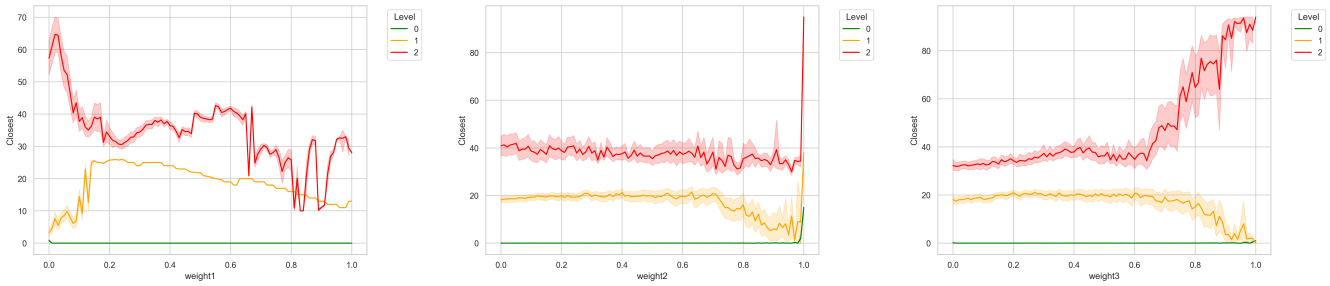


Figure 8: Heatmap of the closest the agent has gotten to the goal



(a) Weight 1 vs Closest (b) Weight 2 vs Closest (c) Weight 3 vs Closest

Figure 9: Effect of weight parameters on the closest the ball has gotten to the goal across levels

5.4 Time Spent in Simulation

Reduction in time can be beneficial if the termination is the result of a completed level. If a level is not completed, the amount of time can tell us how safe a heuristic is.
Heatmap analysis (See figure 10):
Level 1: a weight 1 value of zero results in more time spent in the simulation, in some cases leading to time-outs. This is due to the lack of goal-based behavior.
Level 2: lower values of weight 1 are associated with longer simulation, meaning that the agents remain on the stage for longer periods of time. Additionally, whether it is a higher weight 2 or 3

does not seem to influence the total simulation time.
Level 3: a combination of low weights 1 and to a lesser extent weight 2 leads to longer simulation time. With a high weight 3 being associated with frequent time-outs. Weight 1 accelerates the termination of simulations through failure, as no weight combinations have completed level 2 (see figure 12).
Trend analysis (see figure 11):
Weight 1 (see figure 11a) exhibits a negative impact on the amount of time spent in the simulation.
Weight 2 (see figure 11b) has a positive effect on time spent in the simulation for levels 0 and 1 and little effect in level 2. The increase in time in level 0 means a less optimal path is taken.
Weight 3 (see figure 11c) has a positive effect on the amount of time spent across all levels. As previously mentioned, however, this can be a negative result.

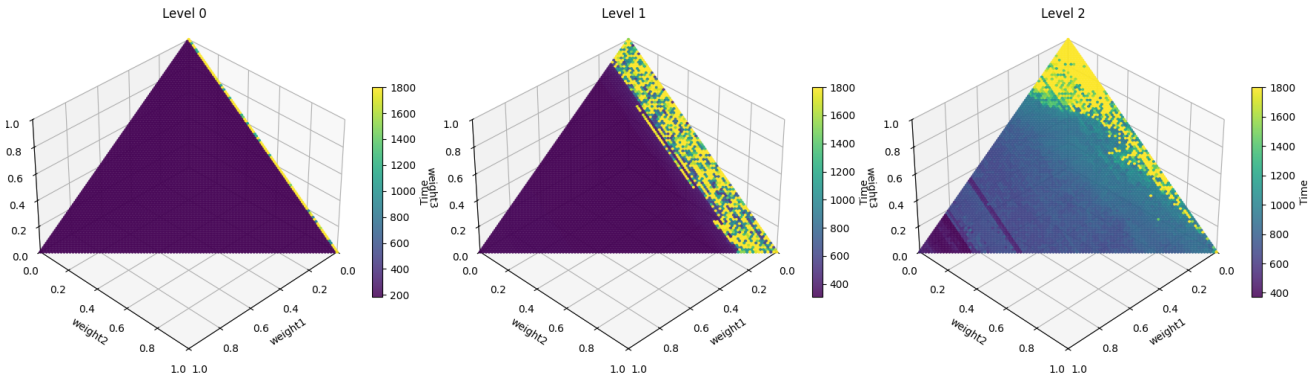
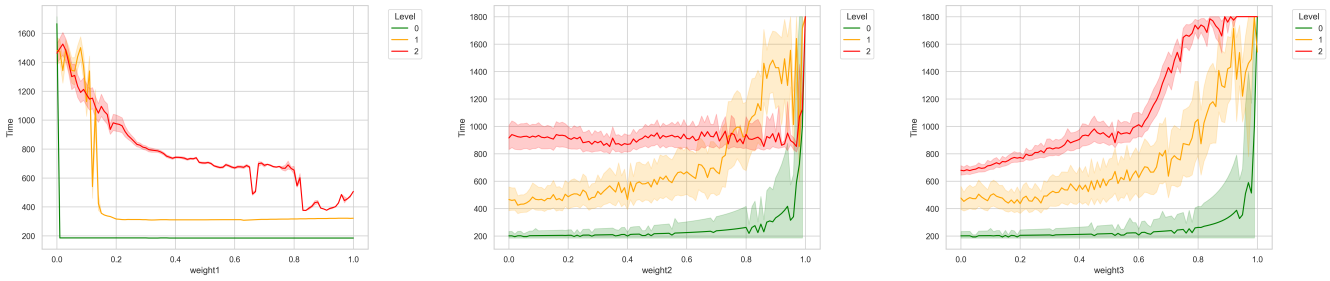


Figure 10: Heatmap of time spent in the simulation



(a) Weight 1 vs Time (b) Weight 2 vs Time (c) Weight 3 vs Time

Figure 11: Effect of weight parameters on the time the ball has spent in levels

5.5 Completion

Level 0: only fails for a very low-weight 1 value. Level 1: completes for some weight combinations where weight 1 ¡ 0.2, works best with a higher weight 2 value. Level 2: Does not ever complete
Heatmap analysis (see figure 10):
Level 0: The agent successfully reaches the goal for nearly all weight combinations, with the exception being a very low weight 1 value.

Level 1: For this level, certain parameter combinations have to be satisfied. Weight 1 should be below approximately 0.2. Performance is best with a higher value of weight 2, which suggests that this component plays a more important role for this specific level.
Level 2: The agent never completed this level under all tested weight configurations. Suggesting that the tested heuristics or weight combinations prove insufficient.

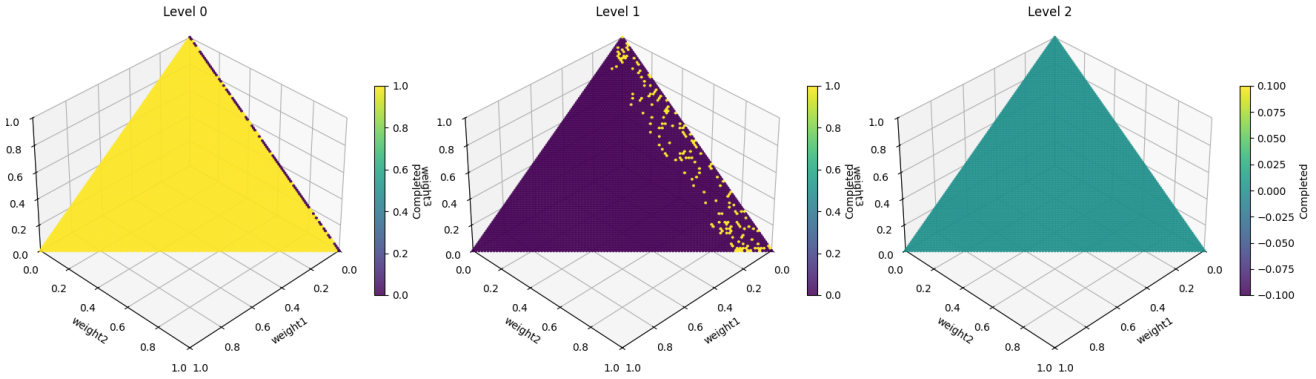


Figure 12: Heatmap of successful and unsuccessful weight combinations

5.6 Baseline Results

When comparing the tables for the weighted heuristics (see table 1, 2, and 3) with the result of the baseline methods (see section 3.3.5), we can see that improvements have been made with certain weight configurations.
Although level 2 is completed by the fine control baseline, the optimal weight combination outperforms it in terms of time. The established fine-control baseline outperforms the weight-based heuristics in terms of total distance.

Table 4: Baseline, Straightforward

Level	Closest	Termination	Total Distance	Time	Completed
0	0	0	14	185	1
1	12	21	95	326	0
2	65	117	290	583	0

Table 5: Baseline, Fine control

Level	Closest	Termination	Total Distance	Time	Completed
0	0	0	14	185	1
1	0	0	122	1228	1
2	45	69	235	719	0

5.7 Optimal Weight Configurations

Each level has a different optimal weight combination; on completed levels, heuristics show their optimal weights in a different way from incomplete levels.
For completed levels, we want to minimize the amount of time the agent spends in simulation. This occurs for the weight configuration:

$$(weight1 = 0.13,\ weight2 = 0.43,\ weigth3 = 0.44)$$

Which has a combined time of 799 frames in levels 0 and 1 (see table 6).

Table 6: Optimal weight configuration for levels 0 and 1

Level	Closest	Termination	Total Distance	Time	Completed
0	0	1	14	186	1
1	0	1	115	799	1
2	41	59	228	941	0

For incomplete levels, we can minimize the "closest" value and secondly the "Distance at termination" value. By minimizing these values, we find the optimal weight configuration for level 2.

$$(weight1 = 0.77,\ weight2 = 0.02,\ weigth3 = 0.21)$$

Which has a closest value of 1 and a distance at termination value of 45 (see table 7).

Table 7: Optimal weight configuration for level 2

Level	Closest	Termination	Total Distance	Time	Completed
0	0	1	14	185	1
1	17	21	88	316	0
2	1	45	257	711	0

6 Discussion

This chapter explains the outcomes of the experiments carried out in the previous section (see section 5) and interprets the results in the context of the research question (see section 1.1). The primary objective of the research was to develop a heuristic approach for a goal-oriented agent using weighted goal- and safety-oriented heuristics, and investigate if this approach is suitable for autonomous agent navigation in a continuous, physics-based environment inspired by Super Monkey Ball. The proposed methods do not rely on planning and are purely reactive.
The discussion will focus on three points.
1. The relation between the weights and navigation performance.
2. The interplay of goal-seeking and safety heuristics in the given environment.
3. Broader implications.

The results indicate that the heuristic blending with the developed heuristics can be a viable approach for navigation, given that the environment does not exceed a barrier of complexity. In more complex and dynamic environments, limitations are found; the environment must be simple to moderately complex, as seen in levels 0 and 1 in the results.

6.1 Interpretation of results

A central takeaway of the research is that no single proposed heuristic has the best performance across all levels. Optimal performance is highly dependent on the level geometry and assigned weights.
This highlights that navigation in the specified environment cannot be solved solely using purely the proposed goal-seeking or safety heuristics alone.
Goal-Oriented and Safety-oriented trade-offs:
As expected, across all experiments, the goal-seeking heuristic proved to be essential for reaching the goal. The weight assigned to the goal-seeking heuristic had to be non-zero for the agent to reliably reach the goal on levels 0 and 1. A weight of zero caused limited movement and often caused the simulation to result in a time-out.
Level 1 (see figure 2) introduced a gap in the middle of the stage geometry. This level could be solved with a relatively low weight assigned to the goal-seeking heuristic and high weights assigned to the safety-heuristics. As expected, more aggressive goal-seeking behavior resulted in a fall-out.
Level 2 (see figure 3) highlighted a limitation of the proposed heuristics. In order to reach the goal, the agent had to move away from the goal, which is not possible with a non-zero weight assigned to the goal-seeking heuristic. High values of the goal-seeking weight cause the agent to reach closer to the goal; this method failed fast and is only possible through the verticality of the geometry.
Distance and Time metrics:
Distance-based metric proved insight into the agent's behavior, in completed levels, minimized time, and total distance covered correlate with the effectiveness of the heuristic.
As expected, these metrics become more ambiguous in levels without completion; a short total distance, for example, can indicate minimal movement, and a low time value indicates a fall-out.

6.2 Generalization

Although the research is inspired by Super Monkey Ball, it can be generalized to a broader spectrum of continuous physics-driven games and simulations.
This encompasses games that involve rolling, sliding, and inertia-based control, such as marble rolling games and physics-based vehicle racing games.
Blended heuristic can serve as an alternative to full-planning; however, the approach suggested in this research scales poorly with growing environmental complexity.

6.3 Scope

Several limitations can be identified in this research.
To start, the simulation is inspired by Super Monkey Ball and does not perfectly replicate its environment. Simplifications from the original game likely impact the agent's behavior.

The use of grid search in this research assumes that the priorities of the agent will not change throughout the levels.
The agent does not dynamically change its priorities throughout a level.
The set of evaluated heuristics is minimal. This is done to study the effects of blending; it also limits the expressive ability of the agent.

6.4 Future Work

The results of this research indicate several limitations that spawn directions for future research.
Several additional heuristics can be created and blended. Some examples are: Slope awareness, detecting if the agent stays in the same area for a long time, and moving away from the goal if there is no direct path.
The agent can take advantage of reinforcement learning to learn optimal weight configurations or switch between weights to select different priorities.
Weight adaptation can be used to change the priorities of the agent during a simulation. If, for example, the agent is close to an edge, it might prioritize moving away from said edge over moving towards the goal.
If an improved version is created, it will need more complex and diverse levels that introduce new obstacles; for example, moving platforms.

7 Conclusion

In this research, a heuristic-based navigation model is developed for continuous physics-based environments inspired by Super Monkey Ball. Traditional graph-based techniques are complicated in this context due to the nature of the environment.
To address this, the research explored whether a weighted heuristic could allow goal-oriented navigation for an autonomous agent.

7.1 Summary of Findings

The research question asked how, using game AI, an autonomous agent can be created in Super Monkey Ball-like environments.
For these three heuristics were implemented: a goal-oriented heuristic and two safety heuristics that base their result on the control vector and velocity of the agent. By blending these heuristics through weighted sums and evaluating their performance through predefined performance-metrics across levels that increase in difficulty, the effect of each heuristic can be studied.
The results show that through this method, the agent can successfully reach its goal in a moderately complex environment.
Experiments revealed clear limitations; the most complicated level could not be completed under any weight configuration, indicating that the proposed approach is insufficient for more complex environments. While certain weight combinations caused the agent to come close to the goal, this was through the verticality of the level and the agent rolling off the edge towards the goal.

Overall, results agree that different weight configurations influence success rate, distance metrics, and time. The optimal weight configuration of the heuristics is highly dependent on level-design, meaning that no single weight configuration is optimal across all environments.

References

[AK23] Jaafar Ahmed Abdulsaheb and Dheyaa Jasim Kadhim. Classical and heuristic approaches for mobile robot path planning: A survey. *Robotics*, 12(4), 2023.

[ASK15] Zeyad Abd Algfoor, Mohd Shahrizal Sunar, and Hoshang Kolivand. A comprehensive study on pathfinding techniques for robotics and video games. *International Journal of Computer Games Technology*, 2015:1–11, 1 2015.

[BBSL11] Vadim Bulitko, Yngvi Björnsson, Nathan Sturtevant, and Ramon Lawrence. Real-time heuristic search for pathfinding in video games. In *Real-time Heuristic Search for Pathfinding in Video Games*, pages 1–30, 01 2011.

[Chi24] Ilia Chichkanov. Development of a swimming game AI, 12 2024.

[Com22] Super Monkey Ball Community. Category: Beginner stages. https://bananapedia.wiki.gg/wiki/Category:Beginner_Stages, 2022. Accessed: 2026-01-15.

[Fou] Blender Foundation. blender.

[Gee24] GeeksforGeeks. Goalbased AI agents, 7 2024.

[GR03] Sheridan S. Graham R., McCabe H. Pathfinding in computer games. *ARROW@TU Dublin.*, 4, 2003.

[GS06] Ross Graham and Stephen Sheridan. Real-time agent navigation with neural networks for computer games. *Artificial Intelligence*, 2006.

[Haw00] Nick Hawes. Real time goal orientated behaviour for computer game agents. In *GAME-ON*, page 71, 2000.

[Kok05] N. Kokash. An introduction to heuristic algorithms. *Department of Informatics and Telecommunications*, 1, 2005.

[LL19] Petro Liashchynskyi and Pavlo Liashchynskyi. Grid search, random search, genetic algorithm: A big comparison for NAS. *CoRR*, abs/1912.06059, 2019.

[LMP03] Michael Luck, Peter McBurney, and Chris Preist. *Agent technology: enabling next generation computing (a roadmap for agent based computing)*. AgentLink, 2003.

[Pan24] Panda3D Team and Erwin Coumans. Panda3d bullet physics module, 2024. Bullet physics integration for Panda3D, based on the Bullet Physics Library.

[RA99] Craig W. Reynolds and Sony Computer Entertainment America. Steering behaviors for autonomous characters. Technical report, Sony Computer Entertainment America, 1999.

[RM10] Gregorio Romero and Luisa Martinez. *Modelling, simulation and optimization*. BoD – Books on Demand, 2 2010.

[Teaa] The Matplotlib Team. Matplotlib — Visualization with Python.

[Teab] The Pandas Team. pandas - Python Data Analysis Library.

[Tea22] Panda3D Team. Home, 2022.

[TtYt] 09Th and the YABEE team. GitHub - 09th/YABEE: Export models from the Blender to Panda3D game engine.

[VG25] Sangeeth Venu and Muralimohan Gurusamy. A comprehensive review of path planning algorithms for autonomous navigation. *Results in Engineering*, 28:107750, 2025.

[Was24] Michael Waskom. Seaborn: Statistical data visualization, 2024.

[WMMJ10] Ben G. Weber, Peter Mawhorter, Michael Mateas, and Arnav Jhala. Reactive planning idioms for multi-scale game ai. In *Proceedings of the 2010 IEEE Conference on Computational Intelligence and Games*, pages 115–122, 2010.

[YT18] Georgios N. Yannakakis and Julian Togelius. *Artificial intelligence and games*. Springer, 1 2018.