Universiteit Leiden
The Netherlands

Opleiding Informatica

Creating a 3D human-like search algorithm

Erik van der Jagt

Supervisors:
Matthias Müller-Brockhausen & Mike Preuss

BACHELOR THESIS

Leiden Institute of Advanced Computer Science (LIACS)
www.liacs.leidenuniv.nl 21/07/2024

Abstract

A contemporary problem in Game AI is acting indistinguishably from humans, therefore damaging the immersiveness of the game. This thesis tries to tackle this problem by showing that algorithms can be altered to showcase more similarity to humans. The type of game this paper researches is a classic 3D hide and seek game, where the seeker will be the human-like game AI. An environment has been separately made for this study, along with the game mechanics. Within that environment, test subjects (n=14) will play this game across three different hider difficulties. Those game sessions will be logged and analyzed. A human-like algorithm was made and matched against two other commonly used seeker algorithms. The thesis concludes with the finding that the algorithm attempting to resemble human behaviour more closely outperforms the standard ones, proving the concept of creating human-like game AI.

Contents

1 Introduction

1.1 The situation

Most video games include some agent that is artificially controlled. For example, there are the non-playable characters, which are often pre-programmed. Such non-playable characters, or NPCs for short, are a core part of every single game where human interaction is involved. As a game developer, you want these interactions between players and NPCs to be as fluent and engaging as possible. The player should feel like they are engaging with another person, since that improves the immersion and overall quality of the game [RS06].
By our analysis in games, hide and seek games like Hello, Neighbor! [gam18], lack the humanity in their AI, even though it was their main focus during the games development. Therefore, we chose this field in games to research. The essence of hide and seek games is that a player is hiding from a searching NPC. Typically, these NPCs feel static and predictable. This damages the game's immersion, as the player does not feel like they are being hunted by a real person. This thesis will try to find out whether it is possible to create a search algorithm, that plans and searches more humanly than other commonly used seeker algorithms.
The corresponding research question is: "To what extent can 3D search algorithms replicate human behavior, and how can one create such a human-like algorithm?". To answer this question, data will be collected on how people typically search, by letting them play a hide and seek game. That information will then be analyzed and used to create an algorithm that tries to resemble the way the subjects searched. That algorithm, alongside other commonly used algorithms in hide and seek games, will be matched against the human data, calculating a similarity score on different metrics: SSIM, POIs, and RMSE. These scores will then be analyzed to determine which algorithm most effectively replicates human behavior.

1.2 Thesis overview

To answer this research question, this thesis will first revise some related work around this subject in section 2. After that, the methodology section 3 covers how the experiments were set up and how they were executed. Next, in section 4, the results of the experiments are presented, together with some reasoning as to why these results are what they seem to be. Then, the algorithms and analysis techniques are explained in section 5. The results of those algorithms are then presented and discussed in section 6. This thesis concludes with section 7, where this thesis' findings will be summarized and future work options are discussed.
This bachelor thesis has been made under the LIACS institute in Leiden, with primary supervisor M.F.T. Müller-Brockhausen MSc and secondary supervisor Dr. M. Preuss.

2 Related Work

Other researches have also been looking into the field of human-like NPCs, and how they can be created. However, regarding the specific methods this thesis uses, not many work has been done, but there still are some that come quite close. A paper by Kopel et al. (2018) [KH18], was interested in finding a way to implement realistic human behaviour into NPCs. They tried using different algorithms and compared them to see which one performed best. This paper specifically

focuses on the intelligent part of human behaviour, mainly assuming that intelligent behaviour automatically confirms human behaviour. However, this thesis will be more interested on the human behaviour part, and less on the intelligence part. Meaning that this thesis is rather interested in acting humanly rather than acting intelligent.
Another recent article is by Milani et al. (2023) [MJM+23], which is a lot more in the direction of this thesis. In this paper, Milani et al. (2023) tries to create an agent that moves inside of a game in a human-like way. In other words, they tried to create a NPC that navigates around the space as humanly as possible. Their results were interesting, revealing that they actually managed to create an agent that passes the Turing test. This means that a human could not distinguish a NPC from an actual player. The essence of this paper is in a broad sense what this thesis will try to produce. The main difference is that this thesis will focus on a search and planning algorithm, rather than a navigation algorithm.

3 Methodologies

3.1 Experimental setup

3.1.1 The scenario

The first major step was the initial design of the experiment. In order to help immerse in the experiment as much as possible, sketching a simple (real-world) story behind the experiment and environment was considered to be beneficial for this research.
The following real-world scenario was created: A human participates in a scientific experiment where researchers have discovered an autonomous robot, which seems to be highly intelligent and is capable of executing a lot of different tasks. However, the scientists only just discovered this robot, and since the original creator has vanished together with all the documentation, the scientists know close to nothing about its capabilities and intentions. The participant is the first person to do an experiment with the robot and the scientists determined that a good first experiment would be hide and seek. It will take place inside a house and when the participant finds the robot, they will be greatly rewarded.

Now let us briefly look at the individual design choices within this scenario. First and foremost, the mysterious robot was chosen because of its unpredictability. Since the participant does not know anything about the entity, it is forced to learn and understand its movement and possible decision and thinking processes. This way, the participant is forced to actively think and then adapt its movement based on what they learn about the behaviour of the robot. This then stimulates the participants to use and create strategies based on what their understanding is of the robots actions, thus stimulating human behaviour.
Another reason why the unpredictable robot was chosen was to evoke human emotion, that could guide their actions and thought processes. The participant could be curious and try to understand it, or maybe the player is cautious and always keeps an eye out for the sudden actions of the robot. In extremer cases, it could cause irritation, making the participant fast but sloppy in their search. It might even cause fear, causing the participant to regularly escape to a safe place or leave the house entirely. All these factors allow for more human behaviour, as a consequence of the mysteriousness of the robot.
Furthermore, a house was chosen because of its resemblance with the real world. Not only is it a

common place to play hide and seek, but it also has a lot of objects that everyone has inside of their house. Sofas, televisions, tables, chairs, etc. are all objects the participant can correlate to in their own real-world home.

3.1.2 The environment

With the scenario sketched the creation of the environment could commence. This experiment is quite specific so the environment needs to be highly adaptable as well as customizable. The decision has been made to not import an existing environment, but create one from scratch. The environment has been made using the Godot 4 game engine [god24], which was chosen because of previous experience of the author with the tool. As previously mentioned, the environment resembles a generic house, with generic furniture. In figure 1 one can see the blueprint of the house. The house features 4 rooms and a hallway, where each section connects to two other sections, with the exception of the hall, where it also connects to the outside area.

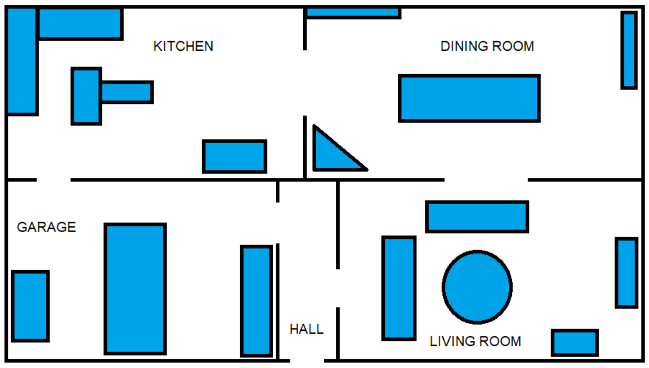
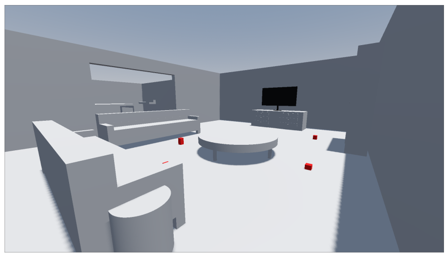


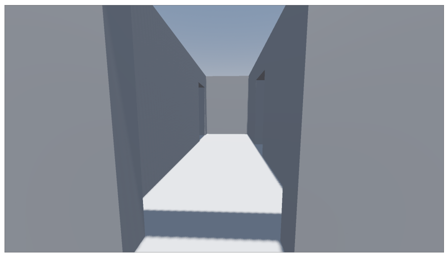
Figure 1: Blueprint of the top view of the house. Each straight black line indicates a wall, and each blue shape represents an unmovable object.

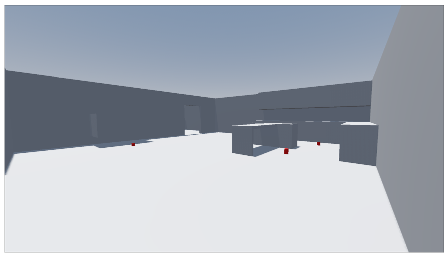
As seen in figure 1 the five sections are: the hall, the living room, the dining room, the kitchen, and the garage. Each room has a set of objects, where some can be used for hiding spots which will be further discussed later on. These objects are objects that are often featured in those types of rooms; the living room has a sofa, the kitchen has a counter, the garage has a car, etc. In figure 2, snapshots have been taken from certain points of view to showcase the exact layout of each room in the experimental environment.
Figure 2 showcases the entire layout of the house. Due to time constraints and the fact that developing a perfect environment is not the ultimate goal of this thesis, the choice was made to make it as practical as possible, by putting only detail into concepts that matter experiment-wise. Therefore, the environment lacks certain attributes like a roof, texturing, sounds, and interactive objects (such as doors).

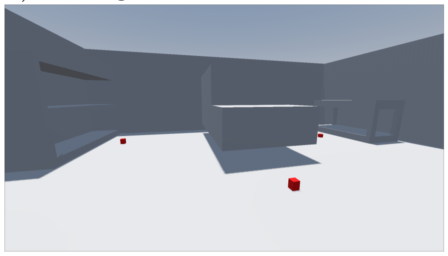
3.1.3 Hide and seek

First let us establish the rules of hide and seek that will be applied in this experiment. The game consist of one hider and one seeker. The goal of the seeker is to find the hider. There is no score


a) Living Room


b) Hallway


c) Kitchen


d) Garage

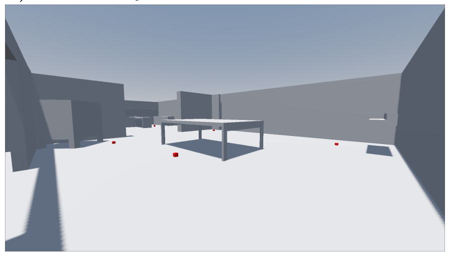

e) Dining Room

Figure 2: Layout of each room in the environment. a) The top left image is the living room, where 2 sofas are on the left, with a table in the middle. Above the table is a dresser with a TV on it, and all the way to the right a closet. b) The middle top image is the hallway where the picture is taken from the outside door looking inside, where there are two doors. c) The top right image is of the kitchen where the is a counter to the right, a small table to the left of it and a dresser on the left. d) The bottom left image shows the garage where there is a shelf on the left, a car in the middle and a workbench on the right. e) Finally the center bottom image is of the dining room, where a furnace is located on the left, in the middle there is a table, behind the table is a curtain, and completely to the right is a piano.

calculation in regards to how well the seeker does. The only relevant factor in this game is that the seeker wins when they find the hider. However, when the hider hides from the seeker for fifteen minutes, the hider is considered the winner. Fifteen minutes was chosen as a result of early testing, and this will be explained in more detail in section 3.2.

The seeker is able to detect the hider in two different ways. The first is to spot the hider while it is walking around, essentially meaning the hider has appeared clearly on the screen. The second is to check a hiding place and if the hider is currently hiding in that place, the seeker is victorious. Finally, there is one self explanatory rule which is that the hider is not allowed to leave the house.

The hider is able to hide within objects. These objects are indicated with a red dot. In order to check whether the hider is currently hiding inside of that object, the seeker must touch that red cube. The color red was chosen because of the contrast with the rest of the environment, so that chances of oversight of these cubes are diminished to a minimum. The red dots are distributed all across the room, because something that is preferably avoided is for the dots to align in a route. This makes it easy for the seeker to define an optimal path and automatically circle the house while always checking all the dots. Not only is this bad for the end results since it leaves little room for movement variation, but it also makes it unfair for the hider.

Since the objects can be collided with, the hider and seeker must find ways around them. This is connected to the argument mentioned earlier. By having objects and obstacles inside rooms, and by placing them well, the seeker has an increased amount of options to overcome that obstacle. For example, since there is a table between where the seeker stands and the red dot it wants to go to, it now has way more options than when there is no table. Instead of going straight to it, the seeker must determine whether they go left or right around the table, or maybe the seeker decided that getting to the point is not worth it anymore because of other variables at play, for instance having the strategy to have at least 2 doorways in sight at all times. All this together has resulted in the current placement of the objects and their corresponding red cubes (figure 1 and 2).

3.1.4 The Hider

As explained in section 3.1.3, the seeker can spot the hider when it moves between hiding places, so the hider needs to have a visible body. This body is shown in figure 3. The body is a dark-tinted capsule, chosen for its semi-clear appearance against the mostly white background. It is thus slightly camouflaged when walking through shadows. An often occurring situation during the development of the environment, was that the hider barely escaped the screen during the search. This causes the player to be unsure whether they spotted the hider. The player is now more curious and cautious, causing more human behaviour, which is beneficial for the results of this experiment.

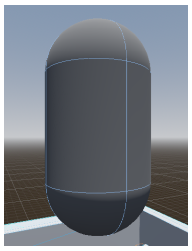


Figure 3: The hider

Since the player is controlling the seeker, the hider needs to have some sort of pre-programmed hiding algorithm. In order to once again enable human behaviour from the test subjects, the algorithm needs to obey two key points. The first is that the algorithm needs to be not too difficult nor too facile, which is self-explanatory (the experiment needs to be doable). The second is that the hider needs to come across as intellectual. This with the thought that players can not walk around mindlessly and expect to catch the hider. If the player realises the algorithm is smart, they need to strategize or at least think about their actions and their consequences, which is what this study wants to obtain from their subjects.

By obeying the previously established rules, the following algorithm was developed. As seen earlier in figure 1 and 2, the house consists of multiple doorways. If the seeker goes through a doorway, the hider will be notified of the seekers location. With that information, the hider will constantly move away from the seekers current location. For example, the seeker moves from the hallway to the living room and the hider is currently hiding under the dining table in the dining room. The hider gets a signal that the seeker is at that location, and will move from their hiding spot towards the dresser in the kitchen. Then, when the player enters the dining room from the living room, a signal will be triggered once again. Since the hider is hiding in the dresser in the kitchen, it will now for example hide under the car in the garage. So in essence, when the player walks the house in circles the hider will constantly move away from the seeker.

Now, the question arises, how does one beat this hider? The player can not walk around aimlessly in circles, checking every hiding spot. They need to develop a strategy or be creative about their movement. This is also how this hider is beaten, by doing something unpredictable. Doing anything but walking around in circles with no intent in mind, will result in success. Suddenly running to the other side of the house, performing small unpredictable movements, or sprinting towards a specific goal are all solutions to catch the hider. The fact that this is the way to beat the hider, is convenient for the ultimate goal of this experiment. The player must at some point develop a unique solution, which the data will reflect, and thus all these solutions can be analyzed and incorporated in the final algorithm. Another nice addition is that the player is incapable of not showing human behaviour. No matter what they do and how, as long as they succeed in the experiment, they have shown some unique strategy or way to find the hider. Even if they do not know how they succeeded themselves, the data will still capture the way they did it.

3.2 Experimental execution

3.2.1 Requirements

First of all, the experiments were run locally. This means that test subjects had to perform the experiment on a device supplied by the researcher, which would be the author in this case. The decision was made not to upload the game onto an online platform. Because of time constraints and the overall load of work it would have taken, it was deemed too much for the scope of this thesis.

Another requirement that the experiment has is that players need to be able to fluently control the seeker. When roaming around the house and looking for the hider, it is crucial that the player needs to have all of its attention towards the finding process, and not be interrupted by how the controls work. The controls of the character has been determined to be WASD for walking where W is forward, A is left, S is backwards, and D is right. Looking around is done by moving the mouse cursor around the screen.

3.2.2 Different Difficulties

Because players needed to play on a supplied device, getting a lot of test subjects proved to be difficult. In order to get as much data as possible from each subject, three difficulties were created for the hider. Now, each test subject can play the game three times, and contribute to three different data sets. These difficulties consist of: the current hiding algorithm (as mentioned in section 3.1.4) being the hardest difficulty, a medium difficulty where the hider only moves away from the player in one direction, and an easy difficulty where the hider always hides behind the counter in the kitchen and does not move at all. The made hiding algorithm was chosen as the hardest difficulty since it proved to be a challenging algorithm. The medium difficulty is an easier version of the hardest one, because it still has the main gist of provoking human behaviour. Then the easiest algorithm was to have a ground truth case, where every player can get used to the environment.

3.2.3 Initial Play Tests

The environment, the hider and the game mechanics in general have all been made from scratch. As for each game, play testing is an important part to see if everything works as expected. Therefore, two participants were chosen to be play testers instead of contributing to the data. In addition to this, gameplay wise some updates can be made so that the player con solely focus on finding the hider and nothing else. This does mean that two less people can perform the actual test where the data will be used for analysis. However, this was determined to be worth it, since it is important that during the actual tests, as little surprises as possible occur.

Apart from removing bugs, these play tests resulted in two major changes: a checking feedback function and a time limit. Showing feedback as to whether a player has checked a hiding spot proved to be important, since it is sometimes hard to determine whether the player walked over one. Players can now know when they are checking a hiding place or not, because of a small pop-up on the screen that says ”checked”. The second is that the players typically do not need longer than a couple of minutes, for any of the difficulties. Therefore, the time limit of fifteen minutes was considered to be more than enough time to find the hider.

3.2.4 Experiment Structure

The structure of the experiments was as follows: the player would sit down behind the device where the environment was set up. Then the scenario sketched in section 3.1.1 was narrated. After this, the player was shown a top view of the environment, and a picture of the hider that they were supposed to find. While these concepts were shown, the hide and seek rules as discussed in section 3.1.3 were explained. After this the player would press the play button whenever they were ready and the experiment started. While the subject was playing, we made notes about the players behaviour. After each round, when the player found the hider, the generated data was copied to a separate folder and given an unique name together with the difficulty of that run. This repeats for each difficulty, from easy to hard. At the end, the subject was done and their thoughts were asked about the written down remarks. Those thought processes were then also recorded. Furthermore, a player was not allowed to discuss their strategies with other test subjects.

Eventually 14 different test subjects did the experiment. The sampling of the test subjects was done with the convenience sampling method, because of the requirements mentioned in 3.2.1. The individuals were all between 21 and 26 years of age. All were proficient with the required controls and have played games in the past with these exact control settings.

During tests, there were some outliers worth discussing. Only once a player did not succeed in finding the hider on the hardest difficulty. The data obtained from this run is however still usable since it still showcases human behaviour and the fact that it also can fail at finding this specific hider. Another, more interesting, outlier is one that deliberately went out of bounds and fell of the map, falling into an endless abyss. While making the environment, this was not something that was taken into account and therefore the run was deemed unusable. The specific player was asked to retry the run and to not go out of bounds again. Both datasets were still saved for both runs, the one that left the environment was not used in the analysis.

3.3 The Data

As previously mentioned in section 3.2.4, data is gathered from the experiments. This data resembles a list of coordinates. These coordinates are two-dimensional, taking only the x and z axis into account, so the length and width excluding the height. The y-axis has not been included, since nor the player or the hider was able to move in that dimension. These coordinates are written into a separate text file, every tenth of a second the player spends inside the game. It writes the exact location the player is at, at that specific time. At the end of the run, the file is closed and has to be copied to an additional folder. There was some discussion about whether to include the angle the player is currently looking at. However, because of the time constraint regarding this thesis and the possible large addition of workload, it was considered to be out of scope for this project.

4 Human Results

4.1 Easy Difficulty

As mentioned in section 3.2.2, the easy difficulty is where the hider simply always hides behind the kitchen counter. Also, as stated in 3.2.4, this is the first difficulty a test subject plays, which is an important precondition to keep in mind. Figure 4 below showcases the three most commonly occurring heat maps within this difficulties results.
The first and most unique situation that occurred multiple times is heat map a) in figure 4. What happens here is that the player has decided to start their search outside of the house. This was either because they had the belief that there would be windows present, or because they were too disoriented when first spawning inside the world. All results of this form look at the left side of the house and all of them walk back after looking around the corner of the house. This situation occurred 15% of the time in the easy results.

Then a more common outcome is heat map b) in figure 4. Here, the player starts their search of the house clockwise, checking all hiding places they come across. This happens in 29% of the cases. The most common result was heat map c) in figure 4, where the player chooses to search the house counter clockwise first, checking all the hiding places they come across. This logically happened in 71% of cases. It makes sense that in most cases the search path is counter clockwise, since the door closest to the entrance is the one to the living room, as best seen in figure 1.

Apart from these common cases, there were not that much unique situations in the result data, or any outliers for that matter. The outlying result was the checking outside the house, but even that situation happened more than once.

There are some interesting patterns that are present in all of these cases. The first one is that the starting position is always a hot point, which is true in every case. This is visible in every heat map in figure 4, where sometimes, as in heat map a), it is not always the hottest point, but a hot point nonetheless. Another pattern is that whenever the player enters a room, they always check every hiding place within that room. They leave the room only when they are confident that they checked every hiding spot in that room. Figure 4 supports this and it also occurs in 92% of the cases, making it a common strategy these test subjects use. It is also the reason why no test subject ever finishes a house search cycle, since it always ends when they enter the kitchen.

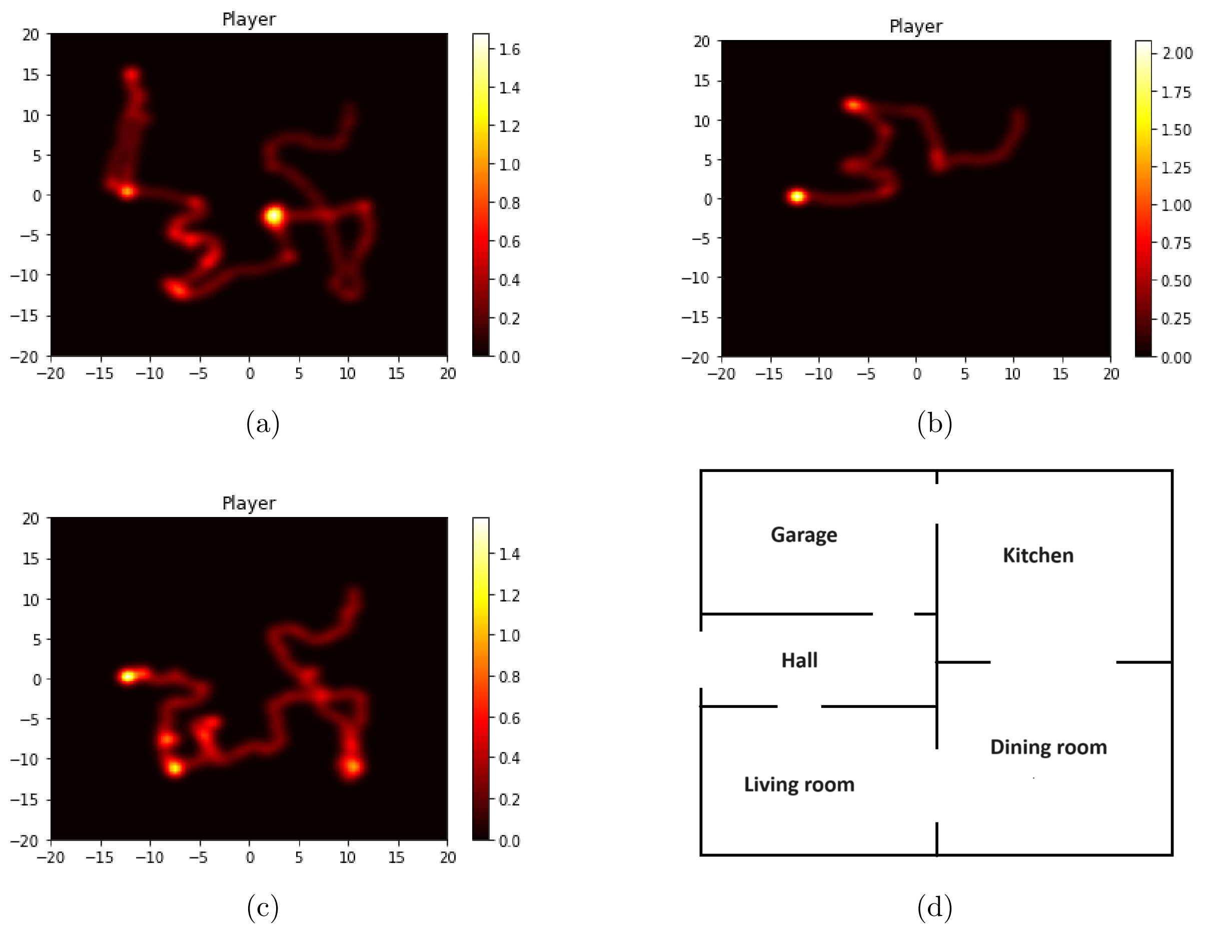
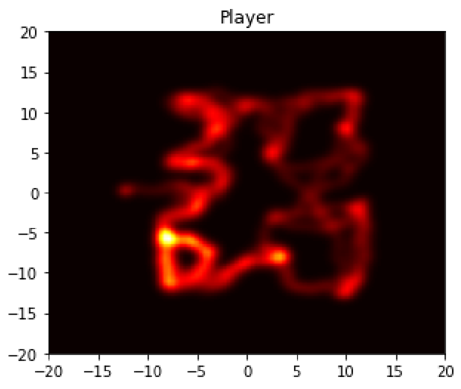


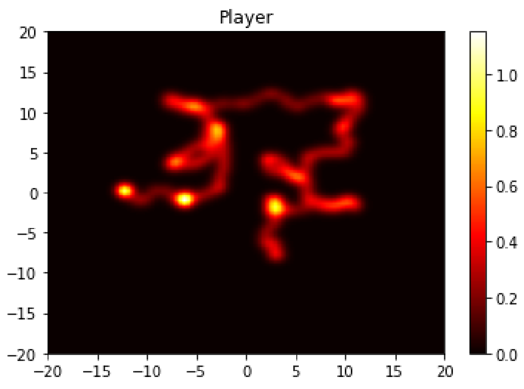
Figure 4: Human results of on the easy difficulty. a) shows a situation where the player first looked partly around the house, and then entered the building. b) shows an example of when a player decided to start searching clockwise. c) is an example of when the player searches the house counter clockwise. d) is a reminder of the blueprint of the house.

4.2 Medium Difficulty

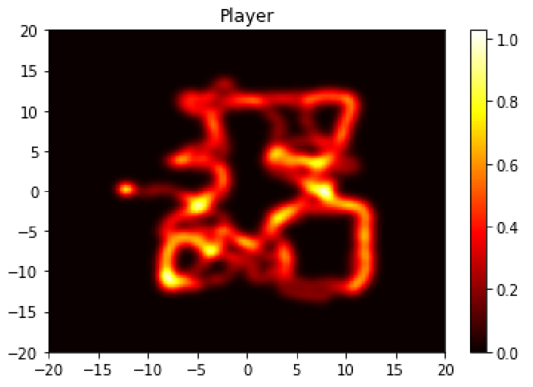
The medium difficulty results are shown in figure 5. In this case there were two common situations and one case that was somewhat different from the rest.



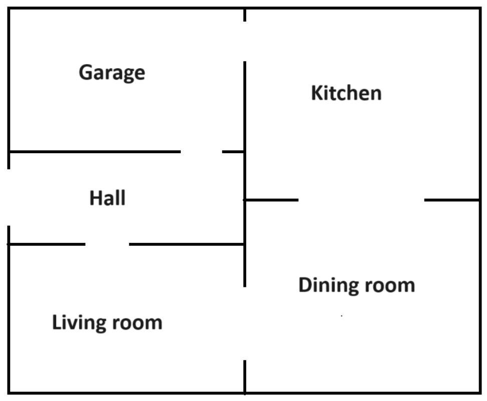
(a)



(b)



(c)



(d)

Figure 5: The results of the medium difficulty experiments. a) shows a typical run, where the player searches counter clockwise and eventually turns around. b) shows a run where the player starts searching clockwise. c) shows a similar case to a), but here the player circles around counter clockwise for a long period of time. d) is a reminder of the blueprint of the house.

An important first notion of these results is that the outside exploration, as seen in a) in figure 4, has not occurred a single time in this difficulty. This makes sense, since the player realized there are no windows in their first run. The player also knows it spawns right before the entrance, and thus knows where to search.

Now for the characteristics of this difficulty. Once again it is about whether the player searches the house clockwise or counter clockwise. The same amount of people, 71% of cases, search counter clockwise. This is likely because the door to the living room is the closest to the entrance. However, it is worth noting that not every player searched the house in the same direction as when they did on the easy difficulty. In other words, the 71% this time around does not convey the same players as the percentage did in the easy difficulty results. Some players decided to switch their searching direction, when going the medium difficulty, in contrast to their easy difficulty run. Situation a) in figure 5 shows the most common heat map, which is the one where the player searches counter clockwise. Every player that searches counter clockwise, eventually finds out that they have to alter their strategy. In this particular example, the player makes one round around the house and on its second round, stops at the table in the dining room. From there, the player walks backwards and starts searching the house clockwise. This is seen because of the hot spot around the table in the living room, marking that the player has been there most often (3 times in this case). The other hotter areas are also the paths that the player has walked twice, for example in the garage.

The other situation is when the player starts searching clockwise, as shown in b) in figure 5. In this case, the player checks every hiding spot again and eventually finds the hider. As explained in section 3.2.2, this hider only moves counter clockwise, so in this case, it does not move at all, allowing these players to always find the hider in the dining room.

The only unique case is heat map c) in figure 5. This is a situation where the player searched the house counter clockwise for around 5 cycles. Normally, it only takes maximally 2 cycles before a player starts looking clockwise. Because of the increased amount of walking, the heat map creates a lot of hot points and a large covered area in general. This data is still usable since it shows that even humans can showcase robot-like behaviour.

4.3 Hard Difficulty

Moving onto the final difficulty, there was no longer a question about in which direction the search went. Rather, by now, every individual had already developed some sort of strategy or idea to find the hider as fast as possible. This also resulted in less cases where the player stands still in the beginning. However, there were still two distinct cases and one outlier as seen in figure 6 below.

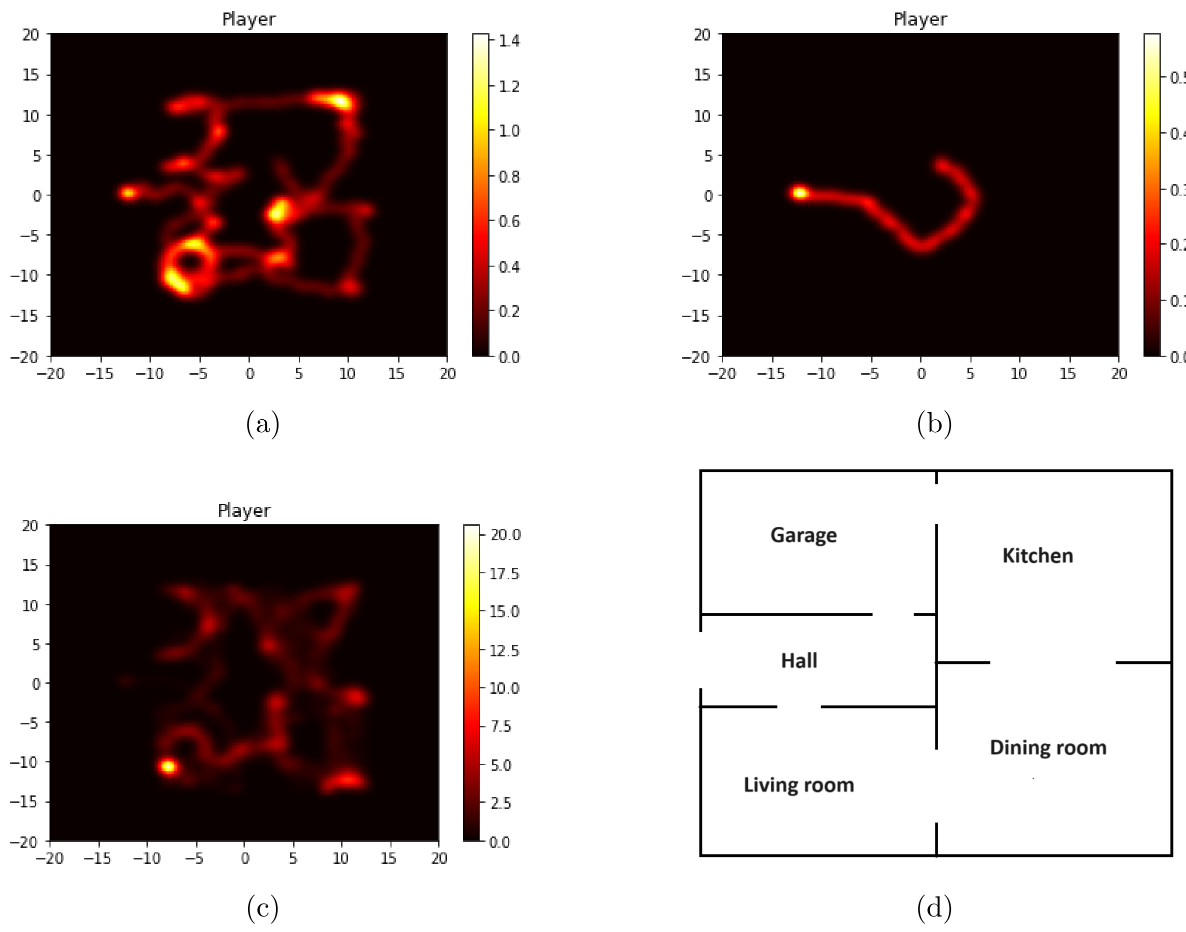


Figure 6: The results of the hard difficulty tests. a) shows the most common heat map structure among the data. b) shows a more unique, but frequent situation. c) is the outlier where the player was unable to catch the hider. d) is a reminder of the blueprint of the house.

Almost none of the cases show exact cycles around the house, checking every hiding place along the way. Even though it still occurs quite often that the player checks all hiding spots when searching a room, an entire cycle of checking every hiding spot in a certain direction is uncommon. As seen in heat map a) in figure 6, the player does not always check every dot in the room when they enter it, which is clearly seen in the kitchen in this specific example. This has to do with the fact that the player discovered in the medium difficulty that walking in cycles is not going to work, especially if the hider is even more intelligent. However, some characteristics still remain, for example the hot circle on the bottom left around the table in the living room. This has to do with the fact that those hiding spots are located rather close to each other, which makes it an efficient choice to always check them all, whenever one of them is checked. Actually in 91% of cases similar to heat map a), hot points are located around the living room table, marking that a core part of the test subjects strategy.

As explained in section 3.1.4, this hider is beaten by doing something unexpected. This is exactly what is also observed in the experiments. Even though not always every individual was aware of the fact that they suddenly made an unexpected move, it always led to finding the seeker. Heat map a) in figure 6 does not show this very well, but b) does. Actually, in 21% of all the cases, such a heat map, or one similar, was generated. This question mark like shape shows that the seeker apparently immediately ran through the living room and through the dining room to immediately start looking in the kitchen. In this case, the hider was hiding in the dining room, and moved to the kitchen as soon as it noticed the player entering the living room. However, as soon as the player entered the dining room, the hider was still on its way to hide in the kitchen. This makes it so that the hider was unable to move to the garage since the player would have spotted the hider, therefore staying hidden in the kitchen. This is then how the player caught the hider. Making such a sudden move is the way to beat this hider and immediately running to the other side of the house to start looking there is a reoccurring strategy according to the data.

Finally there is the outlier, shown in heat map c) in figure 6. This outlier is the one discussed earlier in section 3.2.4, where the player was unable to find the hider. The player cycled through the house multiple times in multiple directions, without finding the hider. Eventually, the player stood in the corner of the living room for a while and observed their surroundings, in the hope that the hider would make a mistake and accidentally pass by. To no avail, the player ran out of time and the run was terminated. Because of this behaviour, the heat map only shows one hot point, since that was the most found data point in the data set by a mile. However, this is still valuable data, since it shows that not every human is able to beat this specific hider, which is why it is kept in.

5 The Algorithms

5.1 Analysis Techniques

Since the goal of this thesis is to compare algorithms with each other based on how well they resemble human data, some metrics need to be used in order to do that comparison. The better the algorithm is able to generate human data, the more that algorithm acts human. Therefore, these metrics need to be able to state how well two datasets match each other. One dataset will be the human data and the other will be data generated by an algorithm. Since algorithms can act different every run, see section 5.2, they need to be ran multiple times in order to generate enough data to resemble the average performance of that algorithm. This is done by making the algorithm run for 10 runs on each difficulty. Since each algorithm run takes at least as much time as human runs, 10 runs were the most efficient amount for each algorithm. Creating a technique that could run experiments faster was deemed to be too much of a time investment, so therefore these 10 runs will suffice. After these 10 runs on a certain difficulty, every run is compared to every human run on that difficulty. Then, the scores of that specific artificial run against each human run is averaged. This gets repeated for each algorithm run, where eventually all the average scores of each run get averaged over once again. That average will then be the final score for that algorithm for that difficulty.

A good way to showcase how coordinates in a dataset behave, is by showing them as a heat map. Heat maps are graphs that show how common certain data points are within a data set. The more often such a data point occurs, the 'hotter' that place becomes in the heat map. If a data point occurs once, it becomes red. When it occurs even more, it gradually turns more and more white based on how many times that data point occurs. When, in contrast, a data point occurs rarely, it turns darker, and if a data point is not present in the database at all, that point is completely black in the heat map. These heat maps are generated from the data by using the numpy library [HMvdW+20]. The heat maps have bins (resolution of the heat map) equal to 100 by 100. The length and width are both ranging from -20 to 20 on both axes. The center of the house is on the coordinate (0,0) and the house has a width and length of about 25. However, the entire platform underneath the house is 40 by 40. Therefore, the heat map is also 40 by 40, since it is technically possible to walk around the house as well. Furthermore, the heat maps are smoothed using a Gaussian filter from the scipy library [VGO+20]. A Gaussian filter is a type of linear filter that applies a Gaussian function to the data points. It calculates new values for each point in the dataset based on the weighted average of nearby points, with closer points weighted more heavily. This filter thus helps with highlighting important paths and patterns.

SSIM Because heat maps are such a handy way of showcasing the data and its characteristics, the first metric is the most straight forward. SSIM, which is short for the Structural Similarity Index, measures the similarity between two images. SSIM compares two images based on three key components: the first being luminance (l), which measures the similarity in brightness between the two images. It compares the mean intensities of the corresponding pixels in both images. Then, contrast (c), which evaluates the similarity in contrast. It compares the standard deviations of pixel intensities in the images. Lastly, the structure (s), which assesses the similarity in the structure of the images. It compares the normalized pixel intensities after removing the mean intensity and normalizing by the standard deviation. The SSIM index is computed as a weighted combination of these three components:

$$SSIM(x, y) = [l(x, y)]^{\alpha} \cdot [c(x, y)]^{\beta} \cdot [s(x, y)]^{\gamma} \tag{1}$$

In equation 1, α, β, and γ are parameters that control the relative importance of the three components. In this thesis, they are set to 1, giving equal importance to all three. The SSIM value ranges from -1 to 1, where: 1 indicates perfect similarity, so the images are identical, 0 indicates no similarity, and negative values indicate dissimilarity.

Within the python script the library scikit-image [PVG+11] was used to apply this metric. The most important parts this metric covers is the overall similarity of the paths taken, but also the speed at which the hider was found. The reason for this is the following example. Take two paths who are completely different, but did find the hider after 3 seconds. Both of their heat maps do not have a red line on the same place, but do have a lot of uncovered ground, which is all black in the heat map. This results in a high similarity since they both have such a large undiscovered area.

Points of Interest Another metric is keeping track of the most important locations in the house that the test subjects often visit. This is done with the help of the heat maps. The 'hottest' places in the heat map are the most visited coordinates by that player. So, when comparing two heat maps for these points of interest, a list of coordinates is formed from the hottest places of both heat maps. Then, both lists are compared and a score is generated based on how many points of interest overlap. The final returned score is then a number that represents how many points of interest two heat maps have in common.

RMSE RMSE, also called the Root Mean Squared Error, is a metric used to measure the accuracy of two data sets. In this case, it compares algorithm-generated data by calculating the average magnitude of errors between the generated data and the actual (human) values. It is computed by taking the square root of the mean of the squared differences between generated and actual observations. The formula is given in equation 2.

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2} \tag{2}$$

Here, the n is the amount of data points, y_i is the value of a data point from the first data set and $\hat{y}_i$ is a data point from the other. The lower the RMSE value, the better the two data sets are alike. In this case, the RSME can look at how much paths differ from each other. However, this is only applicable for short paths within the data. If one compares two complete data sets with RSME, it will not give a suitable result since a lot more variables are at play than RSME can cover. On top of that, it is relatively time-based. For example, if the player data and the computer-generated data take the exact same path, but the player starts 5 seconds later, the RSME will be high, since the data points do not overlap at all. Instead, it is desired to be a low score since the paths are similar, so it needs to get good feedback. However, this method can still be used in strict cases. The decision was made to calculate the RMSE over the first 20 data points, after the first movement has been made. In essence, it calculates the error over the first moves the player makes compared to the algorithm. This is important information since the first decisions of the player are an important part of their strategy and behaviour they show. As seen in section 4, a major part in the strategy of the test subjects is whether they search clockwise or counter clockwise. This feature is something this method can analyse.

Success Rate Finally, there is one metric that sets a baseline for whether an algorithm is even able to be considered to be analyzed. If an algorithm has such a deviating win rate from the human data, it is considered to be doing something fundamentally wrong. If the difference is that significant, it can already be stated that the algorithm is not suitable for comparison. This thesis is comparing search algorithms after all, and if the algorithm is unable to successfully seek, it is not considerable for this research.

5.2 The Artificial Seekers

As mentioned in section 1 and 3, the human data will be compared to the data that the algorithms produce. This section describes the different artificial players, and how they were constructed. The algorithms consist of a random AI, a predictable AI, and a human AI made with the obtained results from the experiments. The random AI and predictable AI were selected because of their common occurrence in similar hide-and-seek games. A book written about game AI [MF09], covers both the use of random walk agents as well as the different applications of Markov chains. Together with other literature explaining and emphasizing on the usability and applications of random walk and Markov chains [Lov93] [Gag17], the decision was made to include these two as algorithms in this thesis. On top of this, some personal experience has been obtained regarding hide-and-seek games. There, it was also discovered that these two algorithms are commonly used as seeking agents within those games.

5.2.1 The Random AI

First up is the random AI (RAI). As the name suggests it is an agent that moves randomly, incorporating the theory of random walks. Random walks are a mathematical process where an entity, such as a NPC, takes a sequence of steps in random directions, often within a structured environment. Each steps direction is determined probabilistically, meaning that each move is independent of the previous one. In the context of game AI, random walks can be used to simulate exploration or searching behavior, providing a simple yet effective strategy for navigating complex spaces without prerequisite knowledge of paths. This method helps ensure that all areas are eventually explored, which is especially useful in scenarios where the exact location of a target (like a hider) is unknown. This RAI will thus randomly pick a hiding spot to check, and after that, pick another random one, up until it found the hider. Its first hiding spot goal is also set randomly.

5.2.2 The Predictable AI

The predictable AI (PAI) is a slightly more advanced algorithm that checks places one after another. Using the theory of markov chains, which are mathematical models that describe a sequence of possible events where the probability of each event depends only on the state generated from the previous event. In other words, they are stochastic processes with no memory, meaning the future state depends solely on the current state and not on the sequence of events that preceded it. In game AI, Markov chains are used to model decision-making and movement patterns of NPCs, where the likelihood of transitioning from one state to another is determined by predefined probabilities. The PAI uses this theory by choosing its next goal based on which hiding spot is the closest and not yet checked. The first goal hiding spot is also set to be the closest by from the starting position. This process repeats until every node is checked. When every node is checked, the PAI chooses the closest node that was checked the longest ago. In essence, this algorithm circles around the house, checking every node in order, repeating the same cycle over and over again. It searches the house counter clockwise, since the closest hiding spot from the start location is located in the living room.

5.2.3 The Human AI

This section will be dissected into three different segments, one for each difficulty. The decision was made to develop a human AI (HAI) for each difficulty separately, because the behaviour of the players together with the data they generated, where indistinguishable each difficulty. Additionally, there is one feature that all of these algorithms will have. Section 4 shows that in every test, the player almost always has a hot point at the starting position, regardless of difficulty. Therefore, every algorithm will wait 2 seconds before starting their run.

Easy As seen in section 4.1, there are a couple of characteristics found in player data. The fist one is the fact that some players first walk around the outside of the house in 15% of the cases. Therefore, an implementation has been made so that the HAI has a 15% chance to search outside of the house first. Another characteristic is that 71% of the cases search the house clockwise and the rest searches counter clockwise. To accommodate this, the algorithm also has a 71% chance to search counter clockwise, and a 29% chance to search clockwise. One final feature the algorithm will have is how it moves between hiding spots. Like the data shows, the algorithm will check every hiding spot in a room whenever they enter it. Then when it is finished, based on in which direction the HAI is searching, it will enter the next room, doing the same routine. In essence, this algorithm waits 2 seconds before occasionally checking left of the house. Then, it starts searching clockwise or counter clockwise. It will fully search each room in that direction until it finds the hider.

Medium Section 4.2 shows the data analysis of the medium difficulty gathered heat maps. First of all, the data does not show any outside of the house exploring, therefore it will not be incorporated in this algorithm. Something that does stay from the easy HAI is the way it searches rooms, since the players still check every hiding place in each room. Another aspect is once again the two different directions the players take. The distribution is once again 29% for clockwise and 71% for counterclockwise. Therefore, once more, the algorithm will take a direction with a chance based on that distribution. In the clockwise case, the algorithm will find the hider in the dining room in their first cycle, like seen in the data in section 4. However, the counterclockwise case is a lot different. In this case, it differs between individuals how fast they turn around and start looking clockwise. The worst observed case is after 5 house cycles, but this a unique case, thus should only happen very rarely. With this in mind, the following implementation was made. When the player finished their first house cycle, they keep their search direction. From this point, whenever it finishes searching one room, it has a 25% chance to start looking clockwise. It was set to this percentage since it most of the time only took 2 cycles at maximum. This way, the HAI should also statistically only maximally do 2 cycles, before looking the other way. Also, it still is possible that it takes more than 2 cycles, therefore allowing for the outlier in the data to be possible, even though that chance is quite small.

Hard As mentioned in section 4.3, the data is not showing one directional cycles anymore. Additionally, the players seem to no longer check every hiding place when entering a room. Both of these methods present in the previous two iterations of the HAI are thus no longer viable. Before talking about these changes any further, incorporating the small percentage group is the easier part, so let us tackle that first. 21% of cases the player skips the first two rooms immediately and start at the opposite side of the house. This feature can simply be implemented by adding a 21% chance that the HAI immediately runs to the kitchen and start searching there.
Now moving on to the hard part, where the problem arises that the player now never performs an entire cycle of looking through the house, checking every hiding spot. The player typically does not search in a direction anymore and they sometimes do not finish searching the entire room. To accommodate these changes, the structure of the medium algorithm can still be used. When entering a room, the HAI will check every hiding spot within that room. To tackle the direction problem, the implementation has been made that after checking an entire room, the HAI checks another randomly chosen room, instead of the next room in the cycle. In this case, the same room is not able to be checked twice in a row. This leaves us with the not finishing a room aspect. For this, the following solution has been implemented, one that also tackles another problem: after the HAI checks a hiding spot it has a 20% chance to search a random hiding spot, which is called the sudden move. The 20% was chosen by testing different numbers and seeing which one correlates most with the human data. With this sudden move, the HAI not only sometimes does not finish searching its current room, but it also now has an unpredictability factor, needed to beat the hider. This unexpected aspect will be the main human strategy involved in this algorithm, which every individual run also showcases.

Since it is a completely random location and every place has an equal chance of occurring, there is of course still a big chance that the sudden move was not ”sudden” enough. When testing, it occurs often that the HAI searches an hiding spot in an adjacent room rather than one on the other side of the house. Because of this, there is a small chance that the hider is able to win. This covers the situation of the outlier, making it, although highly unlikely, a possibility.

6 Results of the Algorithms

This section shows the performance between the different algorithms; random (RAI), predictable (PAI) and human based (HAI). To best showcase these results, tables have been made, one for each difficulty.

6.1 Easy

Table 1 shows the results of the agents on the easy difficulty.

Easy Difficulty

Metric	RAI	PAI	HAI
Success Rate	100%	100%	100%
SSIM	0.715	0.774	0.776
POI	0.11	0.67	1
RMSE	11.6	12.5	10.6

Table 1: Table that shows the performance of each algorithm on the easy difficulty, based on how well they do on each metric against the human data. SSIM is the similarity of heat maps, POI the average corresponding points of interest, RMSE the root mean square error of the first 20 data points after moving.

The first notice is the success rate, which is 100% in each case. This already gives a good indication that on this difficulty, they all qualify since the human data also shows a 100% win ratio. Therefore, as stated in section 5.1, all of these algorithms have a possibility to show human behaviour somewhere. The next metric, SSIM, already shows deviating results. Although at first these numbers seem close to each other, it is important to keep something in mind. Since the heat map also covers the outside, together with the walls and objects, it already has a standard matching black area. The outside of the house already is around 60% of the image and, together with the walls and objects, 66% is generally already equal, apart from some cases where the outside of the house was searched as well. Because of this, a small difference in SSIM score already corresponds to quite massive actual difference between heat maps. In this example, the difference in SSIM between the RAI and the PAI and HAI, is actually quite enormous, stating that the heat maps generated by the RAI do not correlate to the human data. This is of course the result of the fact that the RAI does not search outside of the house, but also because the RAI has a chance to walk around the house multiple times. The human data, as well as the PAI and HAI finishes their run in the kitchen, always leaving at least one room unentered. Because of this, the HAI and PAI have a close SSIM score. The HAI does have the advantage, because it sometimes searches outside the house, and because it sometimes searches clockwise as well.

The POI results have something to do with that as well. The RAI once again has a low score, since it does not show any strategy and has no interest in certain points or patterns. The PAI on the other hand shows some improvement, where on average 0.67 POIs match between two heat maps. This has to do again with the fact that the PAI also searches the house in a cycle checking every hiding spot in each room, similar to the human data. The HAI, however, tops it all where on average they have 1 matching POI. This is the consequence of the fact that also the HAI searches in a cycle, along with the implemented wait time at the start of the run.

Finally, the RMSE shows a high score on all algorithms. This makes sense because of the data points where the player searched outside of the house, thus going in a complete different direction than any algorithm usually. The differences however, can still tell us something about the direction that the algorithms choose. Surprisingly, the RAI performs better here than the PAI. This does make sense however, since the PAI always searches counterclockwise. The RAI on the other hand chooses a random location, therefore is also able to start their search by taking the door on the left. Once again, the HAI outperforms the rest, which is due to the fact that it is able to look outside first, as well as its ability to start looking left or right.

Overall, the HAI is on top on this difficulty. The PAI comes in second by having a good POI and SSIM score. The RAI is behind the most, scoring quite abysmal on the POI and SSIM. However, it does outperform the PAI on RMSE.

6.2 Medium

Table 2 shows the results of the agents on the medium difficulty.

Medium Difficulty

Metric	RAI	PAI	HAI
Success Rate	100%	0%	100%
SSIM	0.691	Nan	0.72
POI	0.17	Nan	1.11
RMSE	3.5	Nan	3

Table 2: Table that shows the performance of each algorithm on the medium difficulty, based on how well they do on each metric against the human data. SSIM is the similarity of heat maps, POI the average corresponding points of interest, RMSE the root mean square error of the first 20 data points after moving.

This time around there is one big difference in comparison to the easy difficult. Table 2 shows that the PAI was not able to find the hider even once. Therefore, as stated in section 5.1, the algorithm is no longer suitable for comparison with the human data. The reason for its low success rate is of course due to the fact that it aimlessly cycles the house, which is exactly what the hider is good against.

On the other hand, the RAI has stepped up its game. Although, the similarity decreased from the previous difficulty, it is closer to that of the HAI. The reason for this is that this time around, walking multiple cycles of the house is something that occurs in the human data. The reason for the overall decrease in SSIM, is the fact that now a larger part of the house is being searched. Before, the leftover rooms were always black, causing a big spike in similarity. This time a lot more rooms are searched, and there is just generally more data points, causing dissimilar heat maps to be less similar. However, the HAI still performs best, which is again highly likely because it has a structure in searching, whereas the RAI does not.

The POIs are once again a grave difference. Here, the HAI still has the advantage of standing still in the beginning of the run, together with the fact that it has a structure of searching the house gradually. Since it checks every hiding spot in each room, the algorithm also generates POIs around points that they walk over whenever they check all the points. An example is the circle around the table in the living room, mentioned in section 4.3.

Finally, the RMSE seems to be rather close between the two algorithms. It has decreased significantly in contrast to the easy difficulty, because there are no longer players looking outside of the house. Since the RAI randomly picks a starting point among the hiding spots, they have about a 43% chance to start left of the house and a 57% to start right. These numbers come for the fact that there are 6 hiding spots on the left side of the house and 8 on the right (thus $6/14 = 57\%$). These percentages are dissimilar to those found in section 4.2. Since the HAI uses the percentages found in 4.2, they scored a lower RMSE.

Once again, the HAI outperforms on every metric. It scores decently better in SSIM and RSME, but does significantly better on the POI metric. It is however worthy to note that the RAI is not falling behind far. Although it performs worse, it does not perform abysmal, if anything, it performs even better than expected.

6.3 Hard

Table 3 shows the results of the agents on the medium difficulty.

Hard Difficulty

Metric	RAI	PAI	HAI
Success Rate	100%	0%	100%
SSIM	0.69	Nan	0.717
POI	0.23	Nan	0.63
RMSE	3.3	Nan	2.5

Table 3: Table that shows the performance of each algorithm on the hard difficulty, based on how well they do on each metric against the human data. SSIM is the similarity of heat maps, POI the average corresponding points of interest, RMSE the root mean square error of the first 20 data points after moving.

Table 3 shows that like in the result table of the medium difficulty 2, the PAI is unable to locate the hider in any of their runs. Therefore, the algorithm is unqualified to be compared to the human data on this difficulty as well. The reason why both the RAI and the HAI have still such a high success rate is of course because they both have an implemented element of surprised, that which is the weakness of this hider.
Now, looking at the SSIM scores, they have once again decreased for both algorithms. The reason has to do with the fact that, once more, there is a lot more ground covered, since the hider is more difficult to catch. Even though both algorithms also cover more ground due to the difficulty, they also do not alter paths between two nodes. If for example a player moves from the piano to the dining table multiple times within one run, they will likely have a different path each time they do so. The algorithms, on the other hand, do not, which is likely why their SSIM score decreases when more ground is covered. Putting that aside, the main interest point is of course the difference between the SSIM score of these algorithms. Since the difference is almost the same as in the previous difficulty, it is logical to say that the same situation still applies. The HAI still has a structure and for example almost always executes the full living room table cycle mentioned in section 4.3, whereas the RAI does not.

The POIs metric is where something interesting happens. The RAI has had a small increase in contrast to their results in table 2. However, the HAI performs almost twice as bad in contrast to table 2. This has likely to do with the fact that there are a lot less players standing still in the beginning, making that POI already not matching with some human data points. On top of that, as mentioned in section 4.3, some players start at the other side of the house, immediately finding the hider. This then results in 0 POIs for that since there are no data points occurring multiple times. These factors are likely what makes it decrease this much. However, the RAI has increased, while they should also suffer from the 0 POI runs. The increase is likely because of the increased search time. Since the players need cover a lot more ground, they also walk through doors more often. The RAI also walks through doors a significant amount of times since they randomly pick a hiding spot to check. The HAI also does this, but in a way more structured way, which is why it is still performing a lot better than the RAI on this metric.

Finally the RMSE which for the RAI did not change much since their results on the medium difficulty, as seen in table 2. This change has probably to do with the fact that the agent is completely random and therefore this time went a couple more times the same way the players went in contrast to the previous time. This metric decreased for the HAI a bit more, which is probably because of the cases that start on the other side of the house, as previously mention in 4.3. Whenever the chance triggers that the HAI does this exact strategy, its path almost perfectly matches the players path in those cases, causing a RSME close to 0. This is therefore likely what caused the decrease.

Once more, the HAI prevails on every metric. Although the RAI is decent competition, the HAI outperforms on every aspect, across every difficulty, showing its superiority.

7 Conclusions and Further Research

To conclude this thesis, let us briefly reflect on each aspect. First, an environment and experiment have been designed for humans (n=14) to play hide and seek in. Human data sets in the form of x and z coordinates were obtained from these runs. That data was then analyzed to search for characteristics within their behaviour. Three different algorithms were developed, one using the random walks theory, one using the Markov chain process and a final algorithm made based on the found human behaviour characteristics. Those three algorithms were then matched against the human data, and the results were that the human-based algorithm performed better than the other two on SSIM, POI, and RMSE, on every difficulty. With this, the conclusion can be made that it is certainly possible to create a human-like 3D search algorithm, that shows significantly more human behavior in contrast to other commonly used search algorithms in games.

Future work that wants to use this specific environment and setup, could look to improve the environment and its aesthetics, so that it looks more realistic and can therefore obtain more human behaviour in tests. More detailed data can be obtained from the test results, such as the y-axis and the look direction. Also, when using more data, one could try out different and more implementations of the human-based AI, and match them against even more algorithms to showcase its limits. A more straight forward change to this experiment could of course be to increase the sample size of test subjects and algorithm runs.

Other future work that wants to look at the bigger picture, can use the methods and theories discussed in this thesis to develop a more general search algorithm that is not bound to one specific environment.

References

[Gag17] Paul Gagniuc. *Markov Chains: From Theory to Implementation and Experimentation.* 05 2017.

[gam18] Hello, neighbor! Online game, 2018.

[god24] Godot game engine, version 4. 17-06-2024.

[HMvdW$^+$20] Charles R. Harris, K. Jarrod Millman, Stéfan J. van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fernández del Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant. Array programming with NumPy. *Nature*, 585(7825):357–362, September 2020.

[KH18] Marek Kopel and Tomasz Hajas. Implementing ai for non-player characters in 3d video games. In *Intelligent Information and Database Systems: 10th Asian Conference, ACIIDS 2018, Dong Hoi City, Vietnam, March 19-21, 2018, Proceedings, Part I 10*, pages 610–619. Springer, 2018.

[Lov93] László Lovász. Random walks on graphs. *Combinatorics, Paul erdos is eighty*, 2(1-46):4, 1993.

[MF09] Ian Millington and John Funge. *AI for Games.* CRC Press, 2009.

[MJM$^+$23] Stephanie Milani, Arthur Juliani, Ida Momennejad, Raluca Georgescu, Jaroslaw Rzepecki, Alison Shaw, Gavin Costello, Fei Fang, Sam Devlin, and Katja Hofmann. Navigates like me: Understanding how people evaluate human-like ai in video games. In *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems*, CHI '23. ACM, April 2023.

[PVG$^+$11] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[RS06] Mark O. Riedl and Andrew Stern. Believable agents and intelligent story adaptation for interactive storytelling. In Stefan Göbel, Rainer Malkewitz, and Ido Iurgel, editors, *Technologies for Interactive Digital Storytelling and Entertainment*, pages 1–12, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.

[VGO$^+$20] Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, Stéfan J. van der Walt, Matthew Brett, Joshua Wilson, K. Jarrod Millman, Nikolay Mayorov, Andrew R. J. Nelson, Eric Jones, Robert Kern, Eric Larson, C J Carey, İlhan Polat, Yu Feng, Eric W. Moore, Jake VanderPlas, Denis Laxalde, Josef Perktold, Robert Cimrman, Ian Henriksen, E. A. Quintero, Charles R. Harris, Anne M. Archibald, Antônio H. Ribeiro, Fabian Pedregosa, Paul van Mulbregt, and SciPy 1.0 Contributors. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, 17:261–272, 2020.