Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Distributed Computing

Hat Hunters

Bachelor Thesis

Nicholas Ingulfsen

ingulfsn@student.ethz.ch

Distributed Computing Group
Computer Engineering and Networks Laboratory
ETH Zürich

Supervisors:
Manuel Eichelberger
Prof. Dr. Roger Wattenhofer

July 29, 2018

Acknowledgements

I want to thank my supervisor Manuel for his support in the weekly discussions and the testing of the game, that lead to many ideas for the enhancement and the further development of the project. I also appreciate the testing with my friends and family and their constructive feedback, which kept my motivation high for working on more improvements and features for the game.

Abstract

This work describes the concept and the creation of the local multiplayer action game *Hat Hunters*. Hat Hunters is a multiplayer game with a shared screen for two to four players with asymmetric information. Hidden input using gamepads or the keyboard are used to achieve this asymmetry.
The game features self-made assets, including a trailer, and is available from the Steam store [1].

Contents

Chapter 1

Introduction

Sitting lonely in front of the computer and gaming is fun. But sitting with some friends around a screen, playing a local multiplayer game and interacting directly with each other in the game as well as in real life is even more fun. Local multiplayer games however often have some restricted possibilities of showing information because everyone can see what is happening on the whole screen. Nevertheless, it is possible to provide some information only to individual players using their hidden input for example with a gamepad. Giving no response at all to an input of the player however can also lead to confusion for the player itself, being not sure whether a certain action was actually performed. Further, it makes is hard for the players to remember all those actions. However, even by giving visual feedback to the player's inputs, it can be hard for others to identify who performed that action if there are a lot of those actions performed by computer controlled characters to disguise the real player's actions.
This work tries to integrate some of those ideas into a local multiplayer action game.

1.1 Related Work

The idea for using asymmetric information on a shared screen originates from the game *Hidden in Plain Sight* by Adam Spragg [2], which consists of a set of mini games for up to four players. The goal in each of those games is to reach certain objectives while hiding amongst computer controlled characters.
In this project, these core ideas of asymmetric information and hiding within a group of equally looking characters are put into a new setting.

Chapter 2

Game Concept

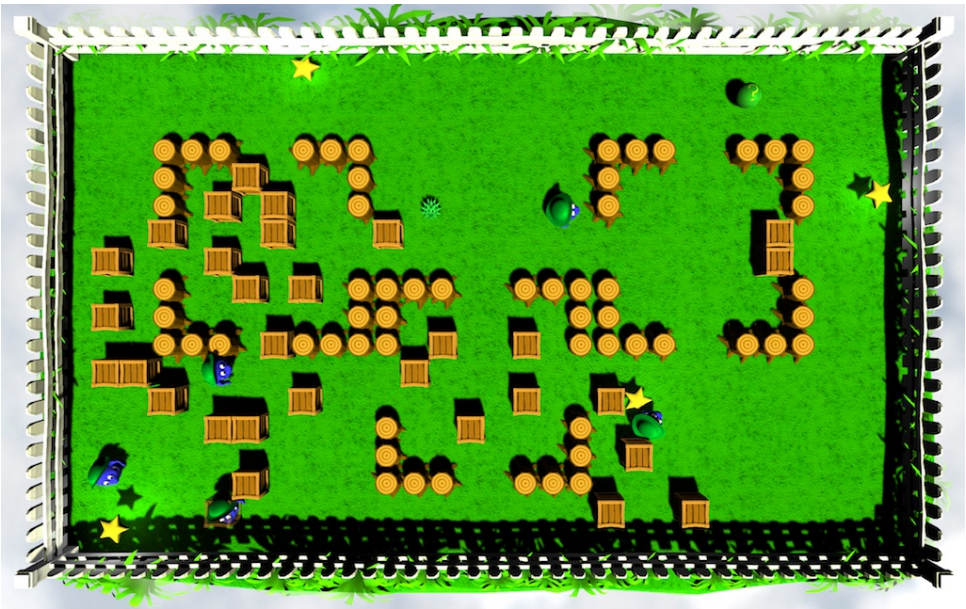

Figure 2.1: FieldTopdown

Hat Hunters is a party game for 2 to 4 players. Each player controls one *Hat Hunter* within a group of identically looking characters which are either controlled by an Artificial Intelligence (AI) or by a real player. The players move on a floating platform and try to achieve their goals without attracting too much attention.

2.1 Goal

The goal of the game is to collect a certain amount of stars. This can be done by either moving towards the randomly spawning stars on the playing field or by stealing them from other characters. The exact amount of stars needed to win

the game depends on the game mode. In the classic mode 8 stars are required.

2.2 Game Elements

2.2.1 Character

The character (Hat Hunter) is a blue creature, which can wear a variety of hats. At the beginning of the game, each character wears the most common hat, the green "Stealth Hat". After a while, when a player gets close to winning, his character changes the hat to the personal one, which reveals his identity to the other players (see Figure 2.2). This rule allows players that are behind to gain an advantage by hiding between identically looking characters.

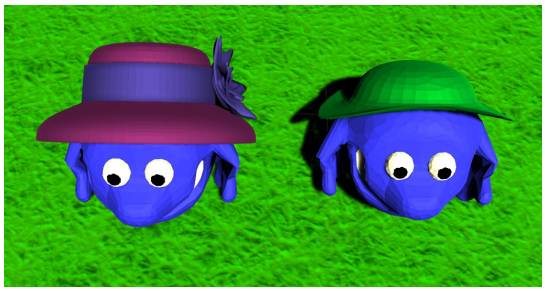

Figure 2.2:
left: personal hat, right: "Stealth Hat"

The Hat Hunter can move freely on the map but cannot cross obstacles or walk through other characters. He can collect stars and items by walking through them. The character can only hold one item at a given time. Any additional item disappears when collected. The Hat Hunter can also place boxes, which can act as an obstacle for others or as a shield for incoming projectiles. Every few seconds, the player gets a new box. The amount of boxes a player can maximally store before placing and the refill rate are determined by the game mode.

2.2.2 Obstacles

There are two types of obstacles in the game: boxes and stumps.
Boxes are placed by players or can be present at the beginning of the match on certain maps. When a player places a box, it spawns at the closest map grid position to the player and stays inactive until no characters occupy that place. Players cannot pass through boxes but can destroy them using various items.
Stumps are obstacles distributed over the field. Their position is defined by the map and can neither be created nor destroyed by players.

2.2.3 Stars

Stars are the key element to winning in the game. The possession of a specific number of stars triggers certain events. When reaching the first event, the

player's hat changes to its personal one. On the second step, the player additionally gets a spotlight around itself and on the final one, the player wins the game.
Stars spawn randomly all over the map in a certain spawn interval. Further, they drop from players who have collected some stars when they get hit by a projectile or an explosion. The lost star then moves away from that player and bounces off obstacles until it gets collected by a Hat Hunter or a certain amount of time has elapsed.

2.2.4 Items

Like stars, items spawn randomly with a given frequency all over the map. When collected, they instantiate to a certain item type, which the Hat Hunter then holds in his hands. The different item types can be seen in Figure 2.3.

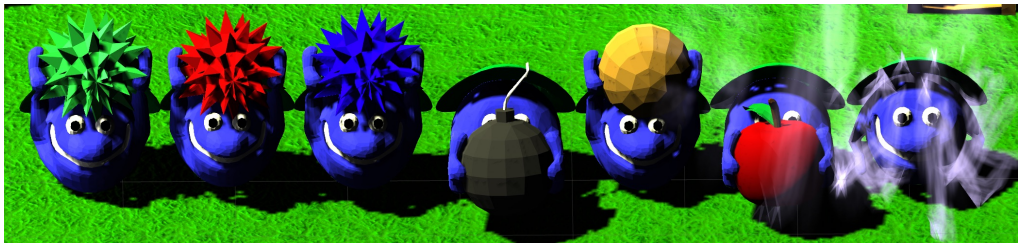

Figure 2.3: Item types from left to right: Green, Red and Blue Projectile, Time Bomb, Catapult Bomb, Apple, Teleport

Green Projectile

This is a simple projectile, which is shot in looking direction and bounces up to 10 times off walls or boxes until it disappears. It can destroy boxes and hit players.

Red Projectile

Unlike the green projectile, the red one does not bounce off walls and can only destroy up to two boxes. This projectile targets the closest Hat Hunter it can find in shooting direction and follows it until it reaches the target or hits a wall.

Blue Projectile

This projectile is the fastest and largest one of the three projectile types in the game. It has about double the hit radius and destroys all boxes along the straight line in which it moves over the field. It hits every player on its path and only

gets destroyed when colliding with a stump or when reaching the border of the map.

Time Bomb

The time bomb can be placed on any free grid-point on the map. After a short delay it detonates and destroys all boxes and hits all players within its radius of impact, which has the size of three units. One unit is the length of a grid tile.

Catapult Bomb

The catapult bomb can be shot over the whole map, which makes it one of the most versatile items in the game, but also one of the most difficult to aim. The longer the player holds the shoot button, the further the bomb will fly. As soon as the bomb lands on the floor it detonates and has the same effect as the time bomb.

Apple

When using this item, the Hat Hunter eats the Apple and as a consequence becomes a giant for a few seconds. When being a giant, the player can walk through boxes and destroy them. The giant is also faster and cannot be blocked by smaller players. However, when walking over smaller players the giant does not hit them.

Teleport

The teleport item allows the player to switch its position with another Hat Hunter. For that, he has to look at a specific character with which he wants to switch places when using the item. The teleportation happens instantaneously and makes it difficult for other players to keep track on who switched places with whom. This adds another hiding mechanism to the game.

2.3 Modes

Previous to starting a game, the players can choose a certain game mode. A game mode consists of a set of rules and parameters for the game, such as how many stars are needed to win the game or how many AI characters are present. It can also be specified which items are allowed in that mode. Additionally, the mode defines if the game is played using the daytime theme (Figure 2.4) or the midnight theme (Figure 2.5). The midnight theme has some dark spots on the

map, which allow players to hide.
There are some predefined modes, but it is possible to create custom modes and import them into the game.

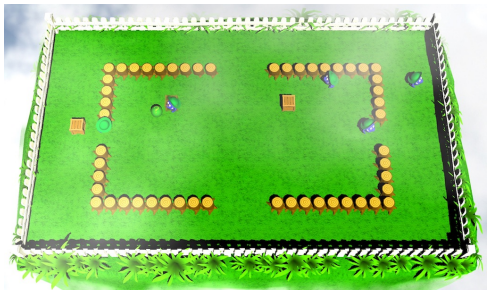

Figure 2.4: Daytime

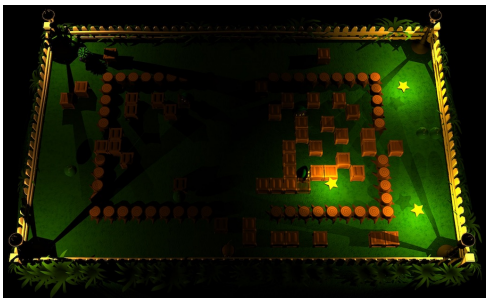

Figure 2.5: Midnight

2.4 Maps

In the menu, it can be chosen on which map the match takes place. The map defines where stumps or boxes are placed on the field. It can also be configured where items and stars can appear or where players spawn at the beginning of the match. Additionally to the predefined maps, it is again possible to create and import self-made ones.

Chapter 3

Implementation

3.1 Unity3D

For the implementation of Hat Hunters, one of the most popular game engines, Unity3D [3], was used. The game engine contains a large set of tools, which help reduce development time a lot. This set of tools can be extended by making use of the assets available on the *Unity Asset Store* [4]. Some of the free assets are also used for this project.
Unity3D has a visual editor, which allows to position static game elements easily in the game world, without having to adjust all positions in code. The main concept in Unity is that of a so called *GameObject*, around which everything in the game evolves. The behaviors or the appearance of *GameObjects* are defined by components, which can be attached to them. By far the most common component is a script, which is written in C# and determines the behavior of that *GameObject*. For visualization there are many predefined components within Unity, such as the *Renderer*, *ParticleEmitter*, *Animator*, *Light* and many more. Those which are frequently used throughout the project are highlighted in the following.

3.2 Game Elements

Most of the elements in the game are created using a construct called *Prefab*. Those are simply *GameObjects* which were created in the visual editor with all their components attached and then stored as an asset, which can then be instantiated at runtime, such as boxes placed by the player.

3.2.1 Field

The game field consists basically of a datastructure that holds information on which elements are currently active on the field. It is divided into a grid, which is used for the placement of objects on the field. For the static elements such as

boxes, stumps or stationary stars there is a two dimensional array of the size of the field representing the grid. Each array element either contains one of those static elements or nothing if that field is free. This array allows for fast access, insertion and deletion of those objects, which are common operations, especially for the physics engine.
All the moving objects, such as players or projectiles, are placed in lists to which objects can be added or removed during gameplay.
The field object is also responsible for populating the field at the beginning of a game with the elements defined by the map and for regularly spawning stars and items.

3.2.2 Hat Hunter

The Hat Hunter is one of the most complex elements in the game. There are basically two kinds of Hat Hunters. The one that is controlled by a player, which has an attached input system and the AI controlled Hat Hunter, which has some custom logic defining its behavior instead. Along with the differences in the control of the Hat Hunter there are slightly different rules for AI controlled characters (bots). Bots cannot win the game, no matter how many stars they collect. Neither do they get a personal hat, but always keep the green "Stealth Hat".

3.2.3 Items

Items can appear in three different forms, spawned, collected and deployed in the game and are treated in each form as a completely different object. When spawned on the field, they are represented by green bouncing question marks, which can be collected by the Hat Hunters.
After being collected, the item turns into a visual representation of that specific item type and is held in the character's hands until he releases it using the shoot button. Which item the player gets depends on his current score compared to the score of the other players. This score is normalized to a value between 0 and 1. The items are arranged in a table (Figure 3.1) from bad to good, into which this score indexes. The lower the score, the better the item. The actual item obtained is then determined randomly by a normal distribution around the player's current score. If only a selection of items is available in a

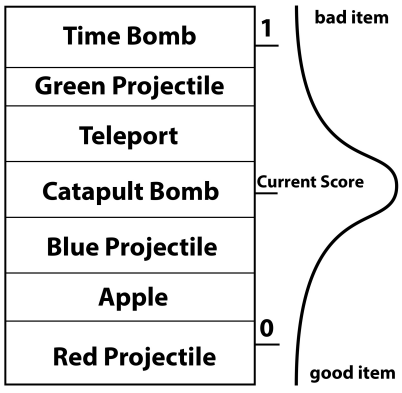

Figure 3.1: Item Distribution

certain game mode, a uniformly random distribution is used over those items. When being deployed, the item representation is removed from the character and the actual item is instantiated on the map and performs its custom logic. The character that releases the item is immune to being hit by that item for a short time period, which prevents the item from immediately hitting its source player.

3.3 Physics

Instead of using the integrated physics engine of Unity, in this project a simple custom physics system was created that fits the needs of the game. One of the main reasons for that decision was to have full control over how the physics objects behave.

Each of the objects in the game, which should collide with others or trigger certain actions when touching another one, has a *PhysicsBody* attached. The *PhysicsBody* defines the shape of the object and whether it should act as a trigger or actually collide with other *PhysicsBodies*.
A moving object, for example a player, checks after every moving step for a collision with all other objects that are nearby. The set of nearby objects contains all moving objects and all stationary objects in the surrounding, which are retrieved from the field array. The surrounding of a player is highlighted in Figure 3.2.

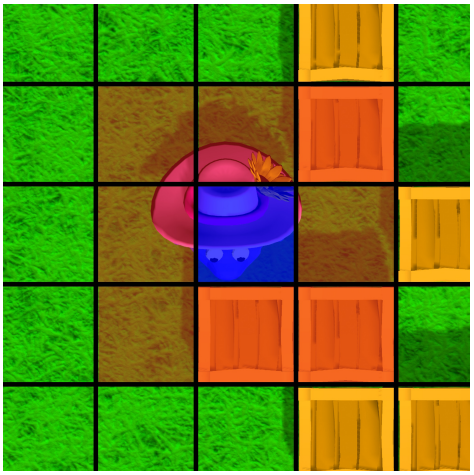

Figure 3.2: Collision Grid

For each of those nearby objects, a collision object is created, which checks if the *PhysicsBodies* actually collide. If a collision takes place, all properties of that collision are calculated, which are necessary for the resolution. For that, the collision depth of the two objects is calculated in the x and y directions as depicted in Figure 3.3. The smaller of the two determines the direction in which the collision should be resolved and the distance that the object needs to be shifted away from the other object.

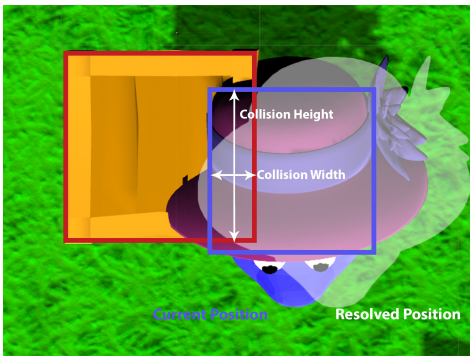

Figure 3.3: Resolving a collision

The relative position of the center points decides whether the resolution direction is up, down, left or right. In the collision, the object that moved into the other one takes the role of the colliding object and the passive one takes the role of the collided object. It does not matter if the passive object is also a moving object, as only one object performs a move step at a time.
After the creation of that collision object, it is handed to the colliding object, which can then decide what to do with it. This decision depends on the types of the two objects participating in the collision and on the current state that the objects are in.
For example, if a player collides with a stump, the player always resolves the collision by moving itself out of the stump. On the other hand, when a projectile collides with a player, it does not resolve the collision but either destroys the projectile and removes a star from the player or simply ignores the collision if the player is immune to that projectile just after it got shot. A detailed collision matrix is shown in Figure 3.4.

	Hat Hunter normal	Hat Hunter giant	Projectile	Time Bomb	Star	Box	Stump
Hat Hunter normal	collide	ignore	hit Hat Hunter	hit Hat Hunter	collect star	collide	collide
Hat Hunter giant	ignore	collide	hit Hat Hunter	hit Hat Hunter	collect star	destroy box	collide
Projectile	hit Hat Hunter	hit Hat Hunter	ignore	ignore	ignore	destroy box (and projectile)	collide (and destroy projectile)
Moving star	collect star	collect star	ignore	ignore	ignore	bounce	bounce

Figure 3.4: Collision Matrix

3.4 AI

The AI describes the non-player characters (NPCs) in the game, which look identically to the real players. The goal of the AI in Hat Hunters is mainly to confuse the players about which character is a real player and therefore allows the players to hide among the NPCs by behaving similarly.
The behavior of the AI is defined as follows:
The AI regularly updates its state, which basically consists of a current target position and a set of possible target positions which have a certain priority assigned. The AI gets notified of each possible target as it appears or disappears. Those targets include items, stationary stars and moving stars, which were lost by a character. Other Hat Hunters are also treated as possible targets. In each update the AI filters the targets in the following way: If the AI already holds an item, it drops all items from the set of possible targets. On the other hand, if the AI currently has no item, it drops the other characters as possible targets. The remaining target positions get filtered by which are reachable on a straight line and if none are left after this, some random target positions around the Hat

Hunter are generated. As the current target position, the position of the closest remaining target is picked.
The movement of the NPCs then simply is steering towards the current target position.
The AI uses the item it holds depending on its type in different situations. For example, projectiles are used when another character is ahead or a bomb is placed if another Hat Hunter is within a small radius around the AI.

3.5 Menu

The menu consists of a *Main Menu*, which is shown prior to the game and allows for player connection (*Join Menu*), mode (*ModeSelection Menu*) and map (*MapSelection Menu*) selection and adjusting game settings (*Settings Menu*).
Further, there are some menus in the game, such as the *Pause Menu* or the *GameOver Menu*.

3.5.1 Joining

The *Join Menu* shows four panels, of which each can represent a connected player. After connecting, a player needs to choose its unique personal hat from the seven available ones. This hat is then used to identify the player during the game when he gets close to winning and to visually represent the winner after a match.

Figure 3.5: Join Menu

When at least two players joined, it is possible to move on to the *ModeSelection Menu*. By pressing the *Start Button* within the *Join Menu*, the *Settings Menu* can be reached.

3.5.2 Modes

In the *ModeSelection Menu* the players are presented a list of game modes. It contains the predefined ones, as well as the custom modes appended at the bottom. For the actively selected mode, a description is shown on the right hand side of the menu.

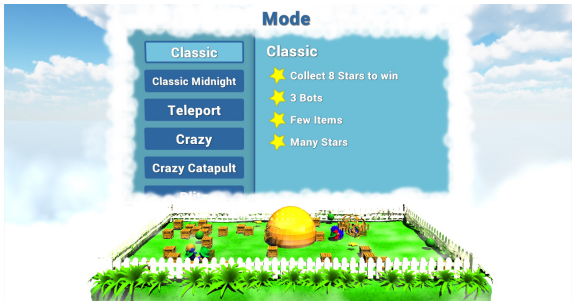

Figure 3.6: Mode Menu

3.5.3 Map

The *MapSelection Menu* is structured similarly to the *ModeSelection Menu* with a list of predefined and custom maps to the left and a preview to the right. The preview is generated from the map data and displays icons for all the game elements and spawn places.

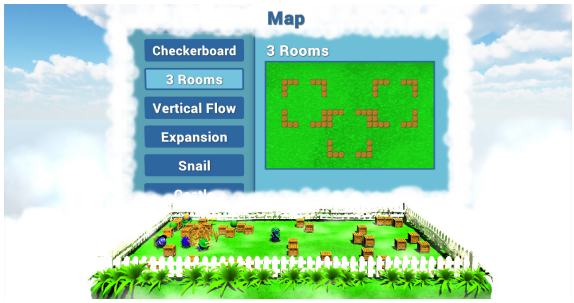

Figure 3.7: Map Menu

3.5.4 Settings

The *Settings Menu* allows to adjust the game's graphics quality, resolution, fullscreen mode, whether audio is enabled and it allows to choose folders from where custom modes and maps are loaded. This menu also shows the controls for the gamepad, as well as for the keyboard inputs.

Figure 3.8: Settings Menu

3.5.5 Game Menus

From within the game, a *Pause Menu* is accessible which also allows to return to the *Main Menu* at any time.

After a completed match, a *GameOver Screen* shows a list of the players ordered by how many stars they reached. This screen has the options to start another round or to return to the *Main Menu*.

3.6 Game Assets

The assets for Hat Hunters are represented by a variety of components attached to the *GameObjects*. In the following sections a brief overview and examples of their creation is given.

3.6.1 3D Models

The 3D Models are all built using the modeling software Blender [5]. The models created in Hat Hunters include the player, hats, items, stars and all the other game elements which have a 3-dimensional appearance. Blender includes a collection of tools that help to generate and position the vertices of a mesh and therefore allows to create such models like the Hat Hunter easily. In Figure 3.9, the mesh of the character is shown. The meshes created in Blender are stored in a format that can be read directly by Unity. This simplifies the modeling and testing workflow a lot, because it is possible to modify the models while running the game.

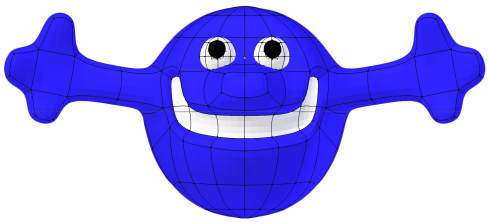

Figure 3.9: Character mesh

3.6.2 Texturing

For most of the 3D elements in the game, textures are not needed, because they only have a few plain colors assigned to their surface. Some exceptions are the playing ground, which has a grass texture, the plants around the hedge and the stumps. The meshes for these objects are unwrapped and overlain by a texture. The tuft of grass for example is simply an arced plane with a partly transparent texture as shown in Figure 3.10.

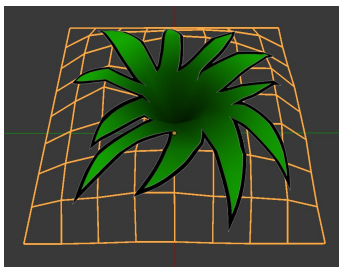

Figure 3.10: Texturing

3.6.3 Rigging

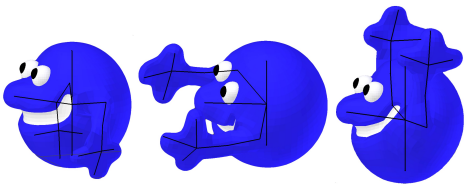

Figure 3.11: Character poses

The only element in the game that has a skeleton (rig) is the player. It is used to deform the player's mesh for animations such as walking, holding an item and throwing a bomb. It is common to model the character in the *T-Pose* as shown in Figure 3.9, which simplifies the rigging. The player skeleton has a hierarchical structure and consists of some bones in the head, which have the bones for the arms and the fingers attached. Using these bones, Blender can add automatic weights for the vertices in the mesh, which define how strongly they are influenced by the movement of those bones. Most of the time however, it is necessary to adjust those weights using the *weight painting tool*, that allows to paint directly on the mesh to specify how strongly it is influenced by each bone.
The rig allows then to set up some poses for the character. A few of them are presented in Figure 3.11.

3.6.4 Animation

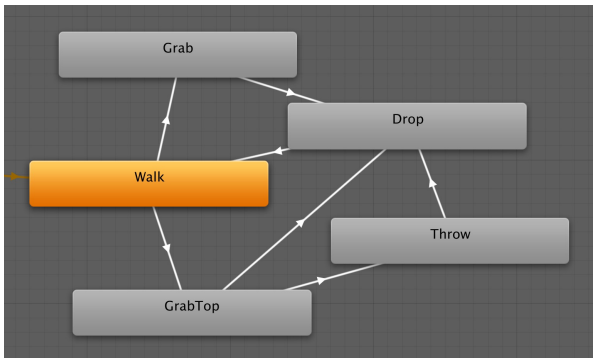

Figure 3.12: Animation state machine

During the game, characters perform some animations when certain events occur, such as picking up an item or throwing a bomb. These are also created within Blender by interpolating the poses described above. Unity allows to import them as well and offers a built-in editor for animation state machines, which describe when objects perform certain animations. Transitions in the state machine can be triggered from a script. This enables for a precise timing of the animations. The state machine in Figure 3.12 shows the player state machine with the *Walk* state being a sub state machine, that blends between walking and standing depending on the player's velocity.

3.6.5 Particle Effects

Particle effects bring a bit more liveliness into the characters and the game as a whole. Unity has a built-in particle editor that allows to create highly customizable effects.
The most dominant particle effects in the game are the clouds, which are floating around or over the field. Further, there are some smaller particle effects when collecting a star or when having the teleport item as shown in Figure 2.3 on the right side.

3.6.6 Audio

The game contains some sound effects for most of the actions and events, as well as an ambient sound for the daytime and the midnight theme. The sound effects come from a website offering free sounds called *freesound.org* [6], from the Steam Asset Store [4] or are self-made using a microphone and the audio editing software *ocenaudio* [7].

Chapter 4

Conclusion And Outlook

4.1 Publishing

To publish the game on the Steam store [8], a number of preparations are necessary.
To support the game for the three platforms macOS, Linux and Windows, especially the gamepad inputs need some extra configuration. To support all these platforms and a variety of different game controllers, where most of them have different mappings, an input management tool, which is freely available on *GitHub* [9], is integrated into the existing one.
For the store page on Steam, some screenshots and a game trailer are necessary. The trailer clips can be created directly inside of Unity, which supports the creation of cutscenes with a tool called *Timeline*. On this *Timeline*, the precise timing of camera flights and game events can be defined. The trailer can be found on the Steam store page of Hat Hunters [1].

4.2 Analytics

Steam provides some basic stats of the game to the developer, like the download numbers or on what operating system the users play. Four weeks after the release on Steam, the download count was at about 2'800.
To get some insight on how the game is played, some basic analytics are implemented. For example, it gets reported which modes, maps and hats are played most often and how many players participate in a match. The hat statistics for the first four weeks is shown in Figure 4.1.

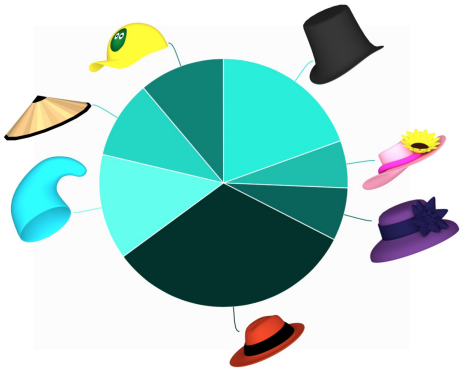

Figure 4.1: Hat popularity

4.3 Feedback

During the development, the game was tested by some friends and after the release, I also got some feedback from players on Steam. Most of the players found the idea nice of keeping the identity of the characters secret. Some even thought, that it should be kept secret a little longer before revealing the hat to keep the excitement level high. According to some Steam players, when using the keyboard, the controls were hard to figure out. To address this issue, a small update was released to clarify these controls by showing keyboard controls in the menus.
The hiding in the crowd of the AI characters turned out to work quite well, as it is, according to user feedback and also from my own experience, often difficult to distinguish the behavior of the AI from the one of the players. However, one issue is, that the consequences are not strong enough of revealing the identity to others. Although the revealed players are a target for others, it is still difficult to hit them with the items or gain an advantage through that in some other way.

4.4 Future Work

Even though the game is already released on Steam it is far from being a closed book. There are lots of opportunities to enhance or extend the game.
The modes and maps currently in the game need more testing and fine-tuning, which is a time consuming task.
There is also a lot of room to extend the game with new modes or new items.
A new sort of collectibles could be added, which improves the players' abilities such as the movement speed or the box placing cooldown.
One could define spots on the map where the player needs to bring its collected stars into safety. This would probably enable more strategies by blocking those spots or waiting around them for other players to pass by, to steal their stars.
There are also ideas to strengthen the punishment of revealing the identity to others by conspicuous behavior. For example, players could mark others as a target for random events such as a lightning strike. This would allow to hit other players only using hidden input and without revealing itself by explicitly throwing a projectile. Another idea is to add an item, which allows to target a player with some kind of cross-hair. If the targeted player is actually a real player, he looses some of his stars and otherwise, the player controlling that cross-hair gets hit by his own item.

Bibliography

[1] : Hat hunters. https://store.steampowered.com/app/874880/Hat_Hunters/

[2] Spragg, A.: Hidden in plain sight. https://store.steampowered.com/app/303590/Hidden_in_Plain_Sight/

[3] : Unity technologies: Unity 3d game engine. http://unity3d.com

[4] : Unity asset store. https://assetstore.unity.com/

[5] : Blender: Free and open source 3d creation suite. http://blender.org

[6] : Free sounds. https://freesound.org

[7] : Ocenaudio: Audio editor. http://www.ocenaudio.com

[8] : Steam: store page. https://store.steampowered.com

[9] : Incontrol on github. https://github.com/pbhogan/InControl