TECHNICAL UNIVERSITY OF DENMARK

02823 INTRODUCTION TO COMPUTER GAME PROTOTYPING

Final report

Egill Ingi Jacobsen s172759
Sigurbjorn Jonsson s172581

Date of submission: December 18th, 2017

Contents

1 Introduction

The game, When the Ninjas stole Christmas, is a 2D platformer about Santa's journey to save Christmas. A group of ninjas had been behaving badly over the last year so they ended on Santa's naughty list. When the ninjas came to this knowledge they decided to ruin Christmas for everyone by stealing all the presents from Santa. Now you as a player have the chance to save Christmas by going all over the world and collect back all the presents that were stolen.

Neither of us had ever programmed in Unity before this course so when it was time to start on our final project we came up with two ideas, 2D platformer and 2D top-down shooter. We divided the ideas between us and over a weekend we worked separately on creating prototypes of these types of games and reported back after the weekend and discussed what game we should do. We decided to go for the 2D platformer as we thought there would be more of free sprite sheets available for that gameplay. Although it was stated from the beginning that it would not be grated for graphics it made the process much more fun to have nice animated sprites rather en squares and circles.

How we ended up with a Christmas game starring ninjas was highly influenced by what sprite sheets we found online as one website had great free assets of Santa, ninjas and, tiles[5].

What motivated us to choose a 2D platformer was first, we both owned Game Boy Color and other game consoles and grew up playing the all-time greats, Super Mario, King Kong and Rayman but also seeing that type of genre is still alive today in example of Ori and the Blind Forest were a simple platformer meets beautiful story and graphics.

2 Detailed Description of the Game

2.1 Game rules

The following is a list of the fundamental rules that apply to the game:

2.1.1 Core Mechanics

1. The player can move by walking, running, jumping and sliding.
2. The player starts a game with six health points.
3. When the player's health points are reduced to zero, the player dies.
4. Enemies can have a different amount of initial health points.
5. When an enemies health points are reduced to zero, the enemy dies.

2.1.2 Player - Enemy Interaction

1. If the player is struck by an enemy attack (ranged or melee), the player loses one health point.
2. If the enemy is struck by a player attack, the enemy loses one health point.
3. If the player enters an enemies line of sight, the enemy will attack the player in different ways depending on the distance between them.

2.1.3 Player - Environment Interaction

1. If the player falls below the lowest platforms of a level, the player loses one health point and is re-spawned at the beginning of the level.
2. The player can walk on top of platforms. Some of these platforms the player can jump through from the side and/or from beneath, but he can never fall through the platform from above.
3. The player can take environmental damage from the following objects only:
   - Bombs
   - Icicles
4. If a player stands on top of a moving platform, the player should move along with the platform automatically. The player still has its own movement and can move internally on top of the platform.

2.2 Player Scoring

1. The player can score points in three different ways:
   (a) Collecting gifts. Each gift collected grants the player 100 points.
   (b) Defeating enemies. For each enemy defeated the player gets 50 points.
   (c) Finding and playing hidden levels.
2. Scores are kept on a per level basis, and on a per-death basis. That means that after each level the player score is aggregated and added to a high score list. After each death, or if the final level is completed, the scores over the entire session (from game start to player death) are aggregated and added to a high score list.
3. If a player ends up with a score that is in the top 5, the player name will be visible on the appropriate high score list.

2.3 Features

2.3.1 Platforms

The platforms are the most important thing in the levels. They allow the player to traverse from start to finish. The platforms come in various sizes and shapes. Each platform has a box collider with a platform effector. The platform effector decides how the player interacts with the platform from different directions. Some platforms can move or disappear.

2.3.2 Gift

They are all over the levels and gives the Player points if he gets them. Sprite was found online[6].

2.3.3 Bomb

Has a box collider that if Player walks into it will trigger Burning animation were the bomb fuse starts to burn and when finished the bomb explodes. Explosion radius is enabled if it explodes and if Player is within the radius he loses one life. Sprites were found online[1][4] and then edited in GimpShop.

2.3.4 Icicle

If Player walks under an icicle a box collider triggers gravity to the icicle and it falls down. If it lands on the Player he loses one life. Sprite was found online[7]

2.3.5 Milk and cookies

Gives Player extra life if he gets it. It moves up and down to indicate it's something the Player wants. Sprite was found online [8].

2.3.6 Chimneys

There are three types of chimneys.

1. Player can finish a level by going down a chimney.

2. Player is blown up in the air by smoke coming out of the chimney.

3. Player can enter bonus level by going down a chimney. It blinks indicating it's special.

We made the sprites for the chimneys ourselves from scratch. The program we used is called GIMP, which stands for GNU Image Manipulation program. GIMP is a free to use photoshop program.

2.4 Technical Challenges

Both of the group members have an extensive experience in programming of some sort, so we were both confident that the task of making a game was well within our capabilities. Even though we have programming experience we only had an introductory experience using Unity. This did not concern us since there are good tutorials available to learn about both the Unity editor and the Unity game engine itself if needed. As the development got further along, we felt that we had got a good handle on the game engine and needed to spend less and less time on tasks of equal size. We wanted to make our code look good so we decided that we wanted to implement state machines for our characters. This is not something we had done before, so we were excited to get started and learn how to implement a state machine with the proper states for the character to move, attack and more.

Since this is the first game either of us develops in Unity we had no experience with setting up animations and animations controllers. By spending some time on that to start with, we quickly became familiar with the process and learned of further functionality that is available in the Unity animator, such as using the animation itself to call functions at the appropriate time.

We also had some technical challenges that are not directly related to the development of the game itself. We wanted to use a library called LibPD to play procedural sounds in the game from Puredata patches. The difficulty with that was that we were running Unity 2017 because of the tilemap it supports. Unity 2017 only supports a 64-bit editor, but the LibPD library only supported at the time a 32-bit editor. We spent a lot of time to try to get the 64-bit version of LibPD to work, but to no avail. We ended up installing Unity 5.6.4 to get 32-bit editor support but that version does not support the tilemap. The reason we wanted to use the tilemap is that it makes decorating the levels easier. The sounds in the game are actually another project that we worked on together in the course *02826 - Sound Design for Digital Media*.

3 Implementation

3.1 Development process

Before development started we decided to approach the game as one would in a studio development, i.e. instead of making every sprite and animation for the game our selves we looked for free assets on the internet and made use of them. This mimics the development in a studio where the art department makes the art and sprites for the game and the game developers integrate the product of that work into the game.

3.2 Inheritance

In early stages of the development, it was decided to break the game down as much as possible. That means that each functionality/behavior in the game has its own script. Where it was possible inheritance was used to simplify the code. As an example of inheritance, a character class was created and the player and enemy classes inherit from the character class. The common behaviors of player and enemies are therefore implemented in the character class.

3.3 Animations

Every animation for the player and the enemy is a list of images that are rendered in a specific order. Each animation is created as a unity animation and they are then set up in an animation controller (or animator) that defines how the flow between animations can occur and when transitions between different states should occur. The animator is then triggered by a script when specific events happen, such as when the player jumps the player script triggers the players jump animation which is then played during the jump.

3.3.1 Animation events

In some cases, the animations could be used to trigger specific events. We used this when a specific event should be triggered when the animation has reached a certain point. All in all, we used this behavior in three places. They are

Throw When a character (the player or an enemy) throws the appropriate object should be spawned and start to move in the correct direction. This should not just happen at any point during the throw animation. It should, and is set to happen at the time where the throw-able object disappears from the animation. This can be used both when throwing on the ground and in the air.

Death When the player dies the current score collected by the player is calculated and displayed on the screen. This should not happen at the exact moment of the player's death. The death animation should finish playing first. To ensure that the animation is played the player death behavior is triggered at the end of the death animation.

3.4 Game Work-flow

The game consists of a few different scenes. There is one scene for the main menu and a scene for the level selection menu. Each level and sub-level has a scene, and there is a scene for the counting of points and another one for viewing high scores. Each scene has a specific functionality and when they are played in the right order the product is a fully playable game. Figure 1 shows the block diagram of the work-flow.

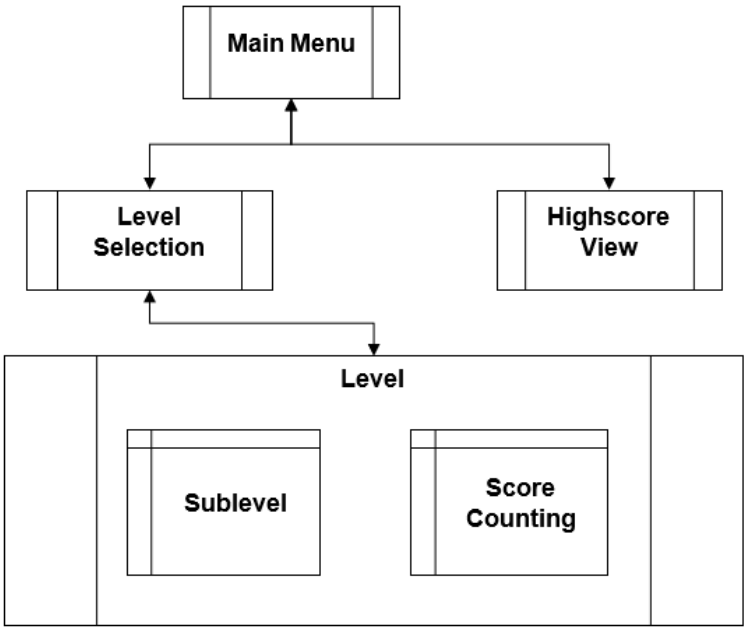

Figure 1: Block diagram of the flow through the game scenes.

3.5 User Interface

The game has five different user interfaces. They are:

- **Main menu interface:** An interface to direct the game towards the high score scene, to the level selection scene or to the instructions scenes.
- **Level selection interface:** This interface contains a few buttons played over an image of a world map. There is a button to go back to the main menu and buttons for each of the game levels. This interface has also a sub-interface, i.e. a small interface that pops up when the user clicks on a button corresponding to a level. This sub-interface contains a high score list for that level and a button to play the level. The unlock button is only here to make the game easier to evaluate and would not be a part of a full release.
- **Level interface:** This is an interface that all the levels have in common, and is defined in the GameManager gameobject. The interface displays the current amount of collected presents in a level and the time, in seconds, since the level was started. There is also a button to go back from the level to the level selection scene.
- **Point counting interface:** The point counting interface is responsible for displaying the calculations of the total score a player has achieved when completing a level or when dying. It also has a button to redirect the user back to the level selection. This interface is loaded in an additive manner onto the level scene, i.e. it is displayed over the level scene itself.

- **High score interface:** This interface displays the top five scores that players have achieved while playing the game. It also contains a button to redirect the user to the main menu scene.

3.6 Level Design

Before we started designing levels, we thought to our selves: "What is the best method to use in level design, that allows us to easily expand the game by making more levels later?". In our minds, the answer to that question is to make easy to use assets, or prefabs as Unity calls it. Those prefabs should be built such that they could function in any Unity scene without any modifications. Because of this desire to make an easily expandable game we started by making a selection of platforms for the player to walk on and other gameobjects with various functionality. After that, we could easily make levels by simply dragging the desired gameobjects into the scene and adjust their positions. In some cases, the prefab gameobject had to adjust a little bit to suit a particular purpose in the scene. For the backgrounds of each level, we found images on the internet that fitted well with the theme of the game. The same thing applies to some of the assets that we use or made for the game. [2, 15, 17, 10, 18, 9, 14, 16, 3, 11, 19]

3.7 Scripts

The scripts we made for the game are numerous. Some of them implement simple functions, but others are quite complex. The most complex and important scripts are described in this section.

3.7.1 Character

The Character class is a parent class for Player and Enemy. What they have in common is what the class has i.e. health, rigidbody, movementSpeed, can be in the air, can throw objects, can take damage and can die.

3.7.2 Player

The player class is the most complicated script. It inherits from the Character class but a player can also slide and go down chimneys. Since the player is moved by the person playing the game we have to handle movement and keyboard inputs. That can be a tricky thing and as we didn't create a state machine for it then there is a function for it that is quite long and could definitely be refactored. In the handle movement function we have to check if the player is in the air, sliding, running, on a moving platform or idling and depending on what to do then we have to play the correct animation each frame.

3.7.3 Enemy

The enemy inherits from the Character class and is implemented with a state machine. This means that there is not only one script that controls the enemy. There is, in fact, one script for each state in the state machine, and then one main script where the enemy is initialized. The states that the enemy can be in are: *Idle*, *Patrol*, *Melee* and *Ranged.* The enemy is in the idle state when it is not moving or attacking. If the enemy is in the patrol state it means that the enemy walks back and forwards along platforms looking for the player. If the player enters the enemies sight the enemy transitions to either the ranged or the melee state depending on the distance from the player. Because the enemy was implemented with a state machine the enemy scripts turned out a lot simpler than the player script.

3.7.4 Platforms

Not all platforms in the game contain a behavior that needs to be scripted. Some platforms only contain a box-collider that allows the player to walk on top of the platform. The only platform that needed a scripted behavior was the moving platform. The moving platform is a platform that moves along certain axes with a constant velocity. After the platform moves a certain range from the starting position the platform turns around and moves back. The implementation of this was fairly simple, although it took a few iterations to find the correct functions in Unity for the platform to work well. The script uses a **Translate** function to change the position of the platform. By adding the player as a child of the platform when the player is on top of it, the player moves automatically with the platform while retaining its own movement.

3.7.5 Game Manager

The Game Manager is a script that is persistent in the game in all scenes. It contains the definition for the user interface that is visible when playing any level in the game. The interface shows the time elapsed since the level started, how many gifts the player has collected and a button to go back to the level selection menu. Since the Game Manager is a persistent object across many scenes, it was decided to use it to store some variables that needed to be retained between scenes. These variables are the player health, the player score and what levels the player has unlocked.

3.7.6 Highscore

The high score in the game is saved to a json file. We use JsonUtility that is built into Unity to achieve this. Looking back the JsonUtility wasn't the best tool to do this since we had do some workarounds to make it work but instead we could have used SimpleJSON[13] which is a more convenient Json parser and a plugin from Unity as well.

- **Highscore:** Contains the structure, the name of the achiever and the score he got, of highscore.json
- **JsonHelper:** A helper script found online[12] but changed to work with our project. FromJson function uses JsonUtility read text string as json and parse it to an object.

ToJson takes the object and serializes it into json string.

- **SaveLoadHighscore:** This script uses the JsonHelper to first load the content from highscore.json and also writes new high scores to the file.

3.7.7 Capture Game

The only sub-level implemented at the moment is a mini-game where the player runs along a platform trying to catch gifts and avoid bombs that fall from the sky. The probability of a gift and a bomb falling is dependent on the difficulty the level is set to. The level difficulty also determines at what rates objects are spawned in at the top of the level, within the camera's viewport.

3.7.8 Level Recap

When the player completes a level or dies the score obtained in the level is showed on screen. The script that does that implements GUI objects for every button, text, and image in the scene. To determine when each GUI object should be displayed is implemented with a co-routine. The co-routine handles the displaying of each object and then it waits for a predetermined amount of time before displaying the next item. At the end of the co-routine, it displays a button so the player can get back to the level selection scene.

3.8 Implementation issues

1. As a part of the game, we made a moving platform. Before we got the platform working we had to try out a few different implementations. The first issue we ran into was that if the rigidbody of the platform was dynamic, then the collision between player and platform was an issue. We got around that by making the rigidbody kinematic. Another issue was how to move the platform and get the player to move reliably with the platform in any direction. We started by using the **MovePosition** function of the platform rigidbody. When we used that we also had to use **MovePosition** on the player rigidbody. That, in turn, introduced a problem with the player jumping and moving on the platform, and the player did not move reliably on the platform. The second iteration tried to add force to the platforms rigidbody. This method is not desirable if the platform movement is to be uniform and is in fact altogether impossible with a kinematic rigidbody. The last iteration and the one that is used in the game is making the player a child of the moving platform gameobject when he is on top of it. The movement is then handled by using the **Translate** function of the moving platform Transform. This implementation works well because the movement of the platform is uniform, and since the player is a child of the moving platform gameobject he automatically moves on top of the platform when the platform moves.

2. In the start when the player changed the direction of movement from left to right or wise verse, we changed the scale of the player gameobject to flip the player sprite around. This introduced problems when the middle of the player was colliding with a platform because when the scale was changed, the player's collider collided with the platforms collider causing the player to shoot up. This was fixed by moving the sprite renderer and animator of the player to child gameobject. When there the change of direction is done by flipping the sprite in the sprite renderer along the x position, and then moving the sprite by a fixed amount along the x position such that it is inside the player collider.

4 Analysis

4.1 Playability

The game we developed does not include any profanities so it should be suitable for children and adults of any age to play it. The gameplay is fairly simple because there are not many buttons that have to be memorized and we tried to make our key-bindings such that it is comfortable to play the game on a keyboard. Another thing that makes the game playable for all ages is that the level design is not very complicated. In some instances, the player needs to focus to find some objects, such as sub-levels, but the progress in the game is not dependent on completing those things. They are purely a mechanic that allows the player to collect extra score to try and get to the high score list.

By implementing a high score system our intention was to encourage players to play the game and try to get the highest score possible. This, in turn, implies that some players will play the game many times to compete both with others for the highest score, and with themselves by trying to get a better score than last time or trying to beat their own high score.

4.2 Feedback from users

Ordinarily, before a game is released it is tested extensively and the feedback from testers is used to improve upon the game. We did not have a lot of time for user testing, but those that tested the game gave us valuable insights into what we could do better. The feedback that we got was:

1. When in the sub-level the movement speed of the player could be higher. This makes it not as hard to catch gifts when they spawn in at opposites sides of the level.

2. Make the jump force of the player a little bit stronger or adjust some of the platforms in the levels because some places were very hard to reach.

3. Change the position of some enemies, add more enemies or remove some enemies because levels were too easy or hard.

5 Conclusion

5.1 Evolution of the game over development time

We started in Unity 2017.3 because we had seen it had a new feature called Tilemaps. With it, you can draw your tiles straight onto a scene like you have a brush. That would have made the platform creation much easier and would have been easier to have creative maps but since we have to use Libpd then we had to go to 32 bit Unity 5.6 because the new Unity only comes in 64 bit. That meant we had to create a few platform prefabs and use them over and over again in our levels.

Other than that our implementation plan kept its course. Maybe that's because we tackled it in somewhat a lean way. We started with platforms, player and an enemy. We looked at it as our MVP and when it was ready and mostly error-free then we started implementing new things on top of it. Starting with a point system with the gifts, followed by more things to hurt the player with bombs and icicles and ending with a high score system to give a user reasons to play the game over and over again.

5.2 What we would do differently

1. **Player state machine:** Currently everything the player does (movement, attack,....) is implemented in one script, with many if-statements to detect what actions the player should take. We would like to re-implement the player in a state machine manner. In our opinion, this is a more clean and elegant way compared to the current implementation.

2. List better out what classes are going to be created before starting implementation because then you get an overview of what is in common and then create parent classes from the beginning because inheritance prevents you from writing the same code in two places.

3. In further development, we would also like to overhaul our scripts because during the development we learned many new things in Unity that we could use to make our scripts better. An example of this is the use of co-routines, which is entirely new to us.

5.3 Future Development

The game developed here is not a big one, but the way we approached the development makes it easy to add more levels to the game and a big part of future development would be additional level design. Although additional levels would expand the game easily, the game would also need more features. Without more features, it would be a risk that the game would be too much repetitive. The following list contains a description of the features we would add to the game with further development.

1. **Player melee attack:** This includes making sprites for a player melee attack, set it up in the game and implement the melee attack in the player script.
2. **Reindeer:** Make a reindeer gameobject. The idea behind this is to make a reindeer gameobject and the animations for it. This gameobject would be used in certain levels instead of the player since the reindeer would have other properties than the player, for example, it would be able to fly.
3. **Elves:** It is well known that elves are widely considered to be Santas little helpers. We have an idea to add elves to the game, and they could play different parts. The elves could, for example, be someone that Santa could save from a cage/imprisonment. This could grant Santa some points, maybe some kind of a power up, or the elve could become Santas companion, traveling with him through the levels helping him save Christmas. Maybe the player could even take control of the elf to complete some tasks only an elf can do.
4. **Additional sublevels/minigames:** Currently there is one type of sublevel in the game, where the player runs around trying to catch gifts that fall from the skies. By making more types of sublevels and/or minigames the playability of the game could be improved, as it would not be as repetitive and introduces more challenges. An idea for a minigame would be to fly around as a reindeer collecting gifts and avoiding hazardous areas while trying to stay ahead of the side of the level which moves forward.
5. **More enemy types:** By more enemies, we are talking about three things.
   (a) We would like to add some enemies that look different to introduce some variety in the gameplay.
   (b) We would like to expand upon the enemy state machine such that every enemy does not behave in the same way and have different difficulties. This makes the enemies less predictable and should hopefully make the game more fun to play.
   (c) We would like to add boss enemies to the game. An example of this would be to add a boss at the end of the final level. The player would then need to win over this boss to win the game. The boss would be different from other enemies. It would have more health points and different attacks.
6. **Icelandic Christmas theme:** Both of the developers of this game come from Iceland. The Icelandic lore about Santa is that there are 13 Santas that are brothers

and live in the mountains. 13 days before Christamas the Santas start coming down from the mountains with gifts for the children, one santa per night. An idea would be to make a version of this game in this spirit. There would be 13 levels, one for each Santa. The Santas sprites would also be different between levels and they would possibly have different attacks that would be tailored to the lore that surrounds them. It would also be possible to extend this further and make multiple worlds, each one tailored to some kind of lore about Santa Claus.

7. **Introduction video:** We thought that instead of having just a static instruction screen where the game mechanics are described, it would be nice to make an introduction video where the theme of the game would be explained. The video could also contain a part where the game mechanics, such as player movement and environment hazards, are explained.

References

[1] Bomb sprite. https://opengameart.org/sites/default/files/Bomb_anim0001.png, 12 2017.

[2] Christmas ball decorations. https://www.thewallpapers.org/photo/21073/Christmas-Balls.jpg, 12 2017.

[3] Christmas house. https://openclipart.org/image/2400px/svg_to_png/188825/ChristmasHouse.png, 12 2017.

[4] Explosion sprite. http://1.bp.blogspot.com/-h4gHvGnPfH0/UmFUg1riZlI/AAAAAAAAAFU/FGgUImTIGbU/s640/explosjon3.png, 12 2017.

[5] Free game assets. https://www.gameart2d.com/freebies.html, 12 2017.

[6] Gift sprite. https://www.shareicon.net/data/2016/02/07/281188_gift_512x512.png, 12 2017.

[7] Icicle sprite. http://www.lisaws.com/uploads/1/9/1/8/1918265/6054389_orig.png, 12 2017.

[8] Milk and cookie sprite. http://vignette1.wikia.nocookie.net/clubpenguin/images/1/17/Clothing_Icons_5392.png/revision/latest?cb=20131219191309, 12 2017.

[9] Ninja headquarters. http://totalplaystation.com/pic.php?file=/media/games/ps2/genjidawnofthesamurai/images/20050517/genji-3.jpg, 12 2017.

[10] Santas sleigh. https://orig00.deviantart.net/f9d6/f/2012/344/0/f/walfas_custom___santa__s_sleigh_by_grayfox5000-d5nl96q.png, 12 2017.

[11] Santas workshop. https://i.pinimg.com/originals/79/48/89/7948895bdd11a71addbf99c4fec74593.jpg, 12 2017.

[12] Serialize and deserialize json and json array in unity. https://stackoverflow.com/questions/36239705/serialize-and-deserialize-json-and-json-array-in-unity/36244111, 12 2017.

[13] Simplejson. http://wiki.unity3d.com/index.php/SimpleJSON, 12 2017.

[14] Snowy forrest. https://ak9.picdn.net/shutterstock/videos/12337829/thumb/1.jpg, 12 2017.

[15] Snowy mountains background. http://weandthecolor.com/wp-content/uploads/2014/11/3-Snowy-mountains-and-polar-lights-from-a-video-directed-by-Scientifantastic-studio-for-Panasonic.jpg, 12 2017.

[16] Snowy waistland. https://static1.squarespace.com/static/50f9c0efe4b09aeef95944a1/t/54b80a6ae4b0c2c1fadf46fc/1421347436139/, 12 2017.

[17] Wooden planks. https://orig00.deviantart.net/1614/f/2009/353/9/5/wood_study_3_by_devin_busha.jpg, 12 2017.

[18] World map. http://www.vectorworldmap.com/vectormaps/vector-world-map-v2.2-blank.gif, 12 2017.

[19] Xmas white background. http://www.pptbackgrounds.org/uploads/xmas-white-backgrounds-wallpapers.jpg, 12 2017.