Toni Minkkinen

Basics of Platform Games

Tradenomi

Tietojenkäsittely

Syksy 2016

KAJAANIN
AMMATTIKORKEAKOULU
UNIVERSITY OF APPLIED SCIENCES

KAJAANIN AMMATTIKORKEAKOULU
UNIVERSITY OF APPLIED SCIENCES

TIIVISTELMÄ

Tekijä(t): Minkkinen Toni

Työn nimi: Basics of Platform Games

Tutkintonimike: Tradenomi, Tietojenkäsittely

Asiasanat: Tasohyppelypeli, pelimekaniikka, pelin kehitys

Tämän opinnäytetyön tavoitteena on jakaa löytämäni tieto tasohyppelypeleistä ja niiden kehittämisestä. Tämä pitää sisällään tietoa tasohyppelyn määritelmistä, historiasta, ominaisuuksista sekä erilaisista tavoista kehittää tasohyppelypelejä.

Opinnäytetyön alussa käyn läpi tasohyppelypelin määritelmät ja alagenret. Sen jälkeen opinnäytetyö sisältää tasohyppelypelien tärkeimmät kehitysaskeleet vuosien saatossa. Määritelmien ja historian jälkeen opinnäytetyö keskittyy tasohyppelypelien kehittämiseen. Lopulta kerron omasta oppimisprojektistani sekä ajatuksistani siihen liittyen.

KAJAANIN AMMATTIKORKEAKOULU
UNIVERSITY OF APPLIED SCIENCES

ABSTRACT

Author(s): Minkkinen Toni

Title of the Publication: Basics of Platform Games

Degree Title: Bachelor of Business Administration, Business Information Technology

Keywords: platformer, game mechanic, game development

The goal of this thesis is to share the information I have found about platform games and developing them. This includes information about the different definitions of platformers, their history, the features related to them and the different ways to develop platform games.

In the beginning of this thesis, I go through definitions of platform games and its sub-genres. Thereafter, the thesis contains the most important steps in the evolution of platformers along the years. After the definitions and history, the thesis focuses on development of platform games. Finally I tell about my own learning project and my thoughts about it.

CONTENTS

1 INTRODUCTION

Platform games have played a significant part in the gaming culture. Some of the most iconic characters in games are the main characters from platform games like Mario and Sonic. Platformers also have played a key part in the evolution of video games and even helped game consoles to thrive. (Racketboy 2011)

This thesis goes through the basic information about platform games, their history and how to develop a platform game with basic features such as double jump and moving platforms. I also introduce my own project, which facilitates some of the techniques and ideas discussed in the theory part of the thesis.

2 PLATFORMERS

2.1 Definition

’Platformers’ or ‘platform games’ are games that mainly revolve around a character controlled by the player, which runs and jumps to avoid obstacles and/or to defeat enemies. Platformers are often classified as a sub-genre of action games, and is considered to be one of the first game genres. Despite being one of the first game genres, platform games have retained their popularity throughout the years. Platformers can be categorized into two distinct types; single screen platformers and scrolling platformers.

Single screen platform games display each level as a single static screen, as seen on figure 1. These games can have multiple levels, which become increasingly difficult as the player progresses. Most of the old arcade platformer games are single screen platformers due to technical limitations, for example, games like Donkey Kong and Burgertime.

Figure 1. The whole level is visible at all times in Offspring Fling (2012).

Scrolling platformers can be identified by the scrolling game screen as the playable character approaches the edge of the game screen. Just like single screen

platformers, scrolling platformers can have multiple increasingly difficult levels and can also feature boss levels. For example Castlevania Sonic the Hedgehog and Super Mario Bros, which can be seen on figure 2, are scrolling platform games. (Klappenbach 2016)

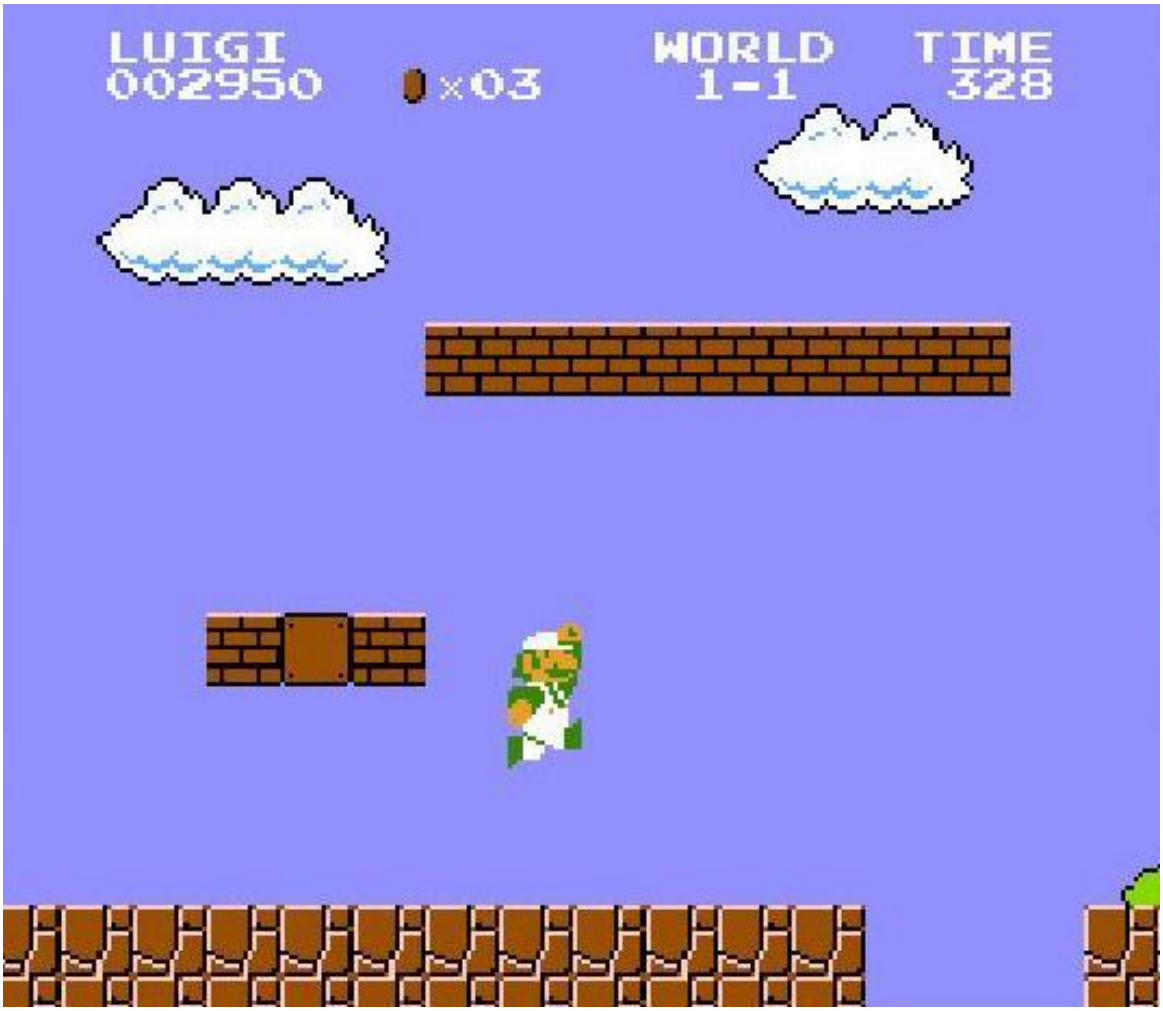

Figure 2. In Super Mario Bros (1985), the camera follows the player.

2.2 History

Platform games as we know them today evolved from a genre of arcade games where the gameplay was centered on climbing ladders. These games took place on a single screen and they did not feature any jumping. The objective was either to climb to the top or to defeat all the enemies on the screen. A game named Space Panic, which was released in the year 1980, was one of these games. It is occasionally credited as the first platform game, but as it does not feature jumping,

it fails to meet the criteria of a modern day platformer. Nonetheless, it is clear that the game has contributed to the birth of platform game genre.

The term ‘platform game’ started to catch on when the press in the UK started to describe the games as platform games during the year 1983. The definition of platformers has developed over the years, and has also spawned multiple sub-genres as mentioned previously. (Electronic Games 1983)

2.2.1 1980s

The game that has more often been acknowledged as the first platformer is the arcade game Donkey Kong published in 1981. The objective of the game is to reach the highest platform on the screen while dodging rolling barrels by jumping over them, see figure 3. Donkey Kong’s popularity gave rise to multiple games which focused on jumping mechanics. Donkey Kong also featured one of the most iconic game characters to be; Mario, who was then known as Jumpman. (Electronic Games 1983)

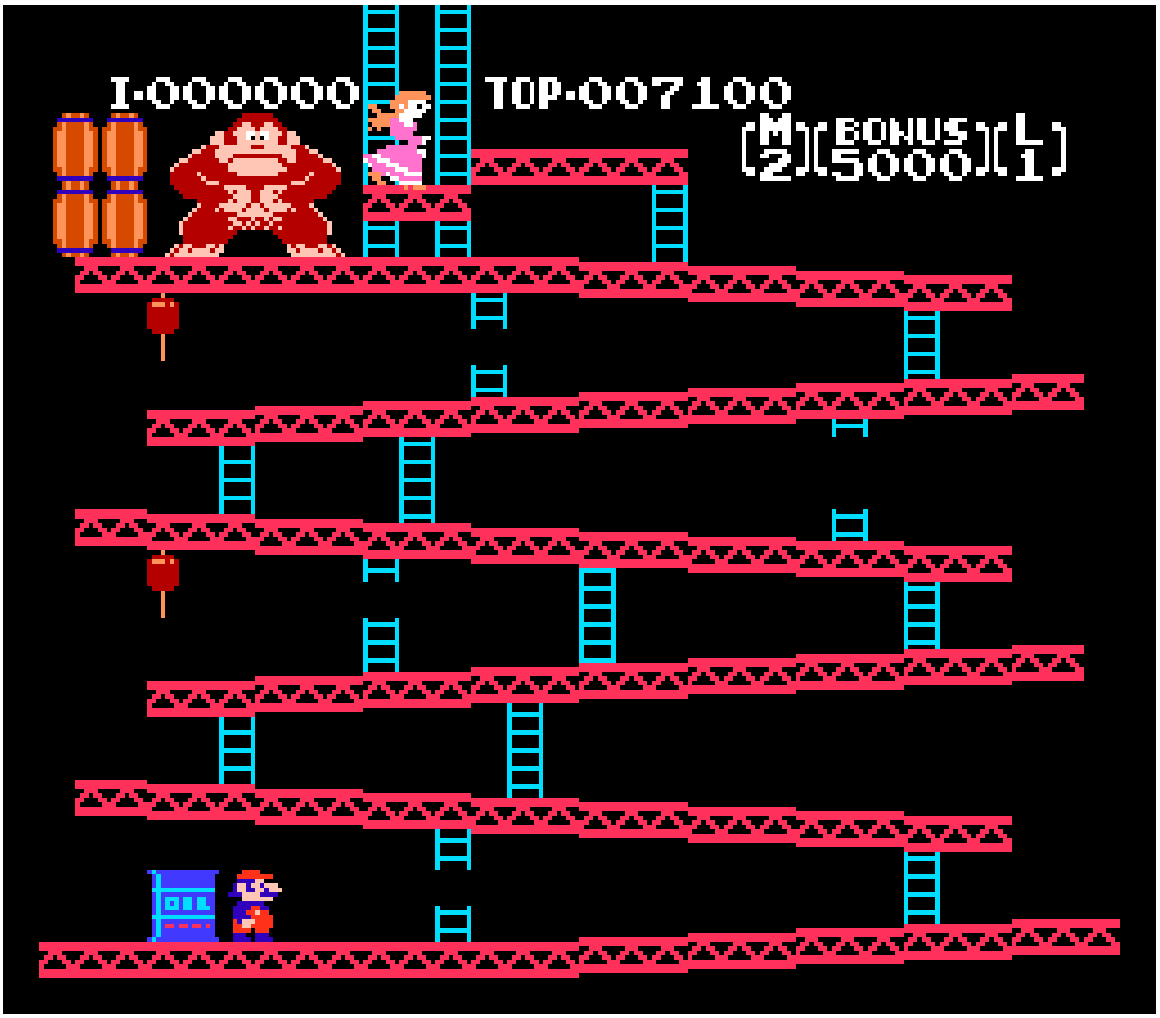

Figure 3. The first platformer: Donkey Kong (1981).

Donkey Kong is also considered to be the very the first **hop and bop** game, which is the most common sub-genre of platformers characterized by their cartoony colorful visuals and mascot characters. The name ‘hop and bop’ derives from the act of jumping on the enemies to defeat them. A Few of the most iconic platformers such as Super Mario Bros and Sonic the Hedgehog also belong to this genre. (Boutros 2006)

Only five months after the release of Donkey Kong, another influential platform game was released named Jump Bug, which merged platforming with shoot ‘em up mechanics. The game had side scrolling levels and the possibility to shoot at the incoming enemies. The following camera in Jump Bug made it possible to create larger levels, so it became a common feature in platform games later on. (Fash 2008, Games Radar 2010)

Jump Bug inspired a genre of platformers called **Run and gun** games that were popularized by Contra and Metal Slug series. These platformers (also known as platform shooters) are platform games that focus on multi-directional shooting, but also feature simple platforming. They have linear levels and have strong ties to their roots as arcade games. **Run and gun** games can feature different power-ups and bosses, and can have a relatively high difficulty. (Retro-Sanctuary)

Another sub-genre of platformers that was born during the early 1980s in a similar fashion. Fusing puzzles and platformers a new genre of platformers was created. These games called **puzzle platformers** use conventional platform game mechanics to drive gameplay where the challenge comes from solving various different puzzles instead of precise movement. Examples of **puzzle platformers** include newer games like Braid and Fez but also older games such as The Lost Vikings. (Racketboy 2011)

In the year 1982 Activision released a game called Pitfall which featured an Indiana Jones type of setting instead of an arbitrary isolated space, making it more realistic and improving the storytelling in platform games. In the year 1984 got a sequel named Pitfall II: Lost Caverns. It was the first platformer to feature checkpoints and was also the first platform game to have an open world, displayed on figure 4, instead of linear levels. (Parish 2014)

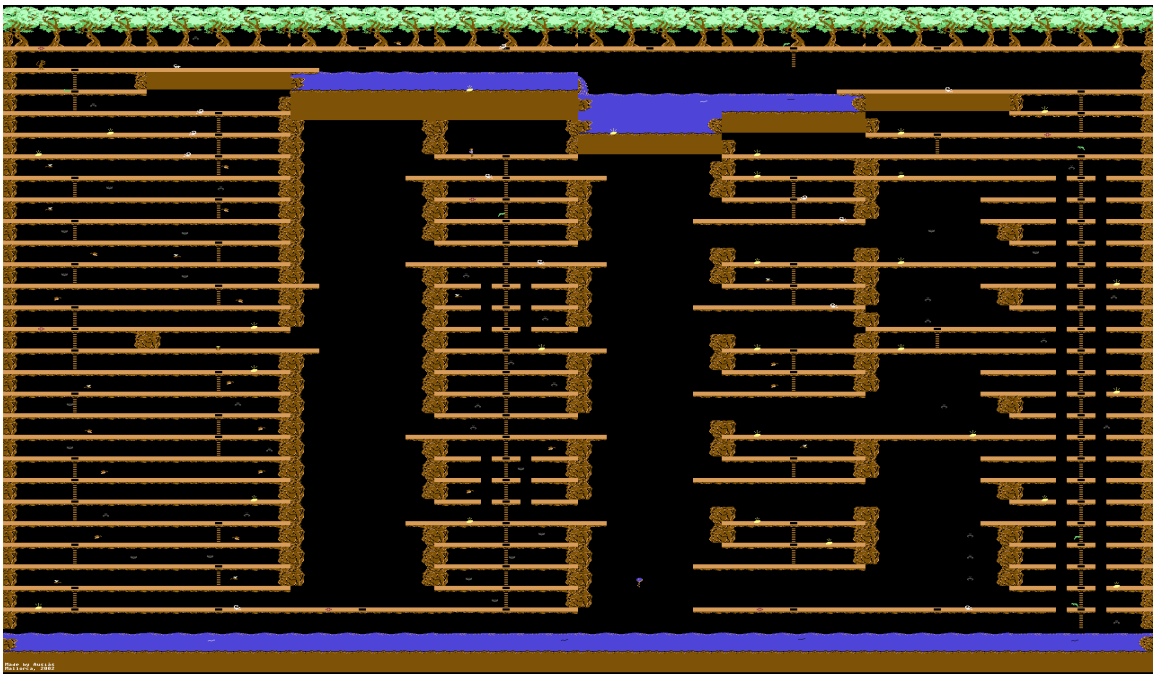

Figure 4. The whole map from Pitfall II (1984).

The concept of open world created a new sub-genre of platform games during the years 1985 and 1986 later referred to as **Metroidvania** games or platform-adventure games. These games fuse platforming elements with various other genres like action-adventure and roleplaying games. Named after two of the genre's most well-known games Metroid and Castlevania, **metroidvania** games feature large open worlds to explore. The game world is often divided into multiple areas that are unlocked as the player gains new skills or items. (Fletcher 2009)

Halfway to 1980s, the console games were dying out. People believed that the gaming consoles were going totally extinct until Super Mario Bros was released in 1985. It managed to turn the tides on the 'video game crash' quickly becoming one of the bestselling video games in history initiating the golden age of platform games. (Cifaldi 2012)

At the end of 1980s, a new sub-genre of platform games came to be. The games were heavily influenced by movies and were later dubbed as **cinematic platformers**. This genre's focus is on fluid lifelike movements and realistic portrayal of characters. As the emphasis is on realism, the games avoid having unnatural physics, and the characters are relatively vulnerable compared to characters in other platform game sub-genres. (Bexander 2014)

The word ‘cinematic’ comes from the usage of a technique called rotoscoping, which is often used when creating the animation sequences for **cinematic platform games**, and from the fact that the character’s animations can’t be interrupted once the action is invoked. The first **cinematic platformer** Prince of Persia was also the first game to use rotoscoping for animation. An example of what the animations looked like can be seen on figure 5. Other good examples of cinematic platformers include Another World, LIMBO and Deadlight. (Bexander 2014, Rybicki 2008)

Figure 5. Rotoscoped animation from Prince of Persia (1989).

2.2.2 1990s

In the wake of Super Mario Bros’ success, the console developers began to think it was crucial for a console’s success to have a flagship platform game with a recognizable mascot character. During the fourth and fifth generation of consoles each one had their own figurehead; Nintendo had Mario, Sega had Sonic the Hedgehog and later on Sony had Crash Bandicoot (Parish)

When Nintendo 64 and Playstation were released, rendering three-dimensional graphics finally became possible. However, the visualization of the third dimension had already begun in the 1980s with **isometric platformers** such as Congo Bongo which was released in the year 1983**. Isometric platformers** are platform games characterized simply by the use of a technique called isometric projection, which allows a two-dimensional environment to appear as three-dimensional. This technique was used especially back in the days when the hardware was not advanced enough for actual polygonal 3D graphics. For example Sonic 3D Blast used isometric projection and pre-rendered 3D sprites to make the game look three-dimensional, as seen on figure 6. (GameZone 2007, Racketboy 2011)

Figure 6. Isometric graphics from Sonic 3D Blast (1996).

In addition to isometric platformers, there were also pseudo 3D games which had a static camera angle and the illusion of three-dimensionality was achieved by deforming two-dimensional sprites. Before actual real-time 3D graphics there were also games which are referred to as 2.5D that used images of pre-rendered 3D objects as sprites and two-dimensional game logic. The first game to meet the criteria of 3D platform game was Alpha Waves, see figure 7, which was already released in the year 1990, though according to Guinness World Records the tittle of first 3D platform game belongs to a game released in the year 1995 named

Jumping Flash. (Fash 2008, GameZone 2007, Gorenfeld 2013, Guinness World Records 2008)

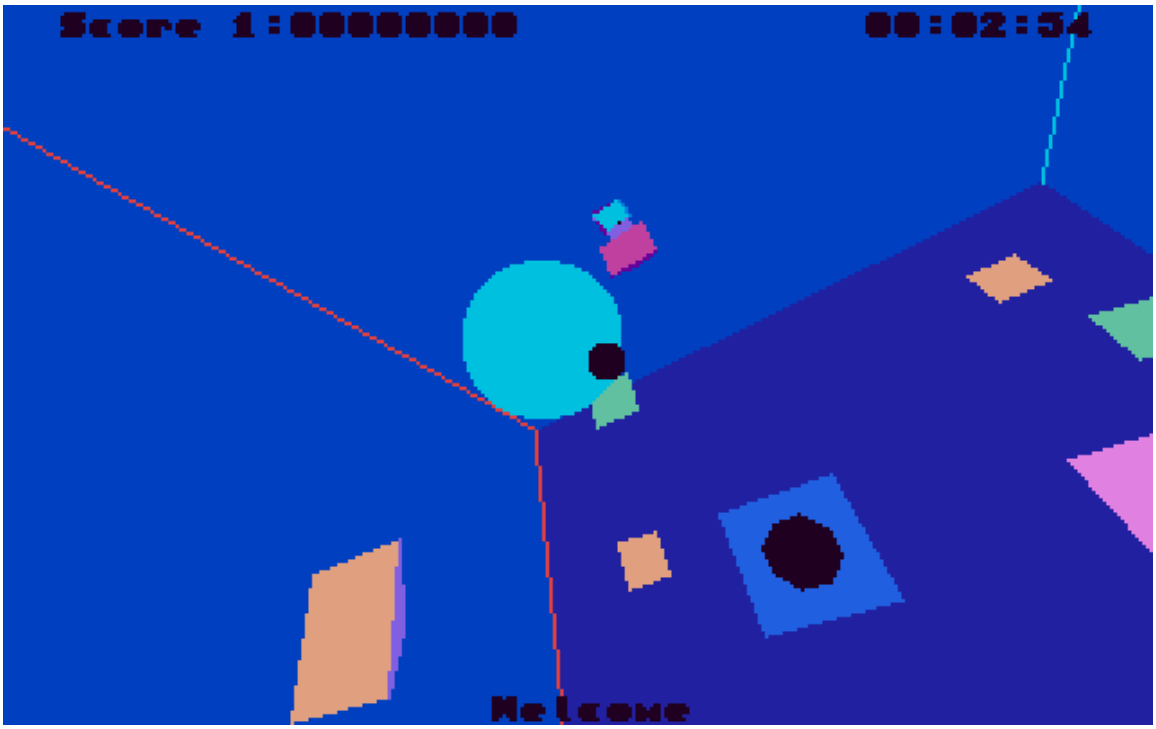

Figure 7. Early 3D graphics from Alpha Waves (1990).

During the first half of 1990s, platform games were still trying to figure out how to approach 3D graphics and gameplay. No-one had ever played a 3D platformer let alone developed one. This was until Super Mario 64 came out in the year 1996. It became a huge success setting the standards for 3D platformer in the years to come. Super Mario 64 featured an open world where the player learnt how to handle the novel aspects like controls and gameplay. The objective based gameplay utilized the 3D world better than a linear approach and gave players the possibility to play through the game without accomplishing all the objectives. The analog in Nintendo 64 controllers gave a solid foundation for 3D games and made moving the character in 3D space more intuitive and accurate. (Parish)

2.2.3 2000s and 2010s

At the beginning of 2000s the popularity of platform games had faded and the golden age had passed. Nonetheless, the genre still remained quite lively. On

consoles and PC, platform games started to fuse into a hybrid of multiple genres where the rich interactivity has been traded with blend of various features. However, on hand-held consoles, platformers have revived the gameplay of the 1980s and 1990s. The games have become more rewarding and easy to play, so they would reel in more players. (Boutros 2006)

The accessibility of game engines and the openness of platforms has given a chance for smaller game companies to market their games for wider audiences. The rise of independent game developers has spawned many new successful platform games like Super Meat Boy and Braid. (Harrel 2013)

The rise of smart phones also brought a whole new platform to the market that was perfect for small independent developers. The unique inputs on mobile phones required a new approach to designing platform games creating a new sub-genre named **Endless Runners**. As the name suggests, **endless runners** are games where a character constantly moves along a never-ending level while trying to avoid different obstacles. Games of this genre are predominantly seen on mobile platforms because they do not require complex inputs. This can be seen in games like Canabalt and Temple Run. (Chong 2015)

3 TECHNOLOGY

3.1 Types of platformers

3.1.1 Tile-based (pure)

Pure tile-based platformer is the easiest one to implement out of the four different types. In pure tile-based platformers, the character's position, as far as the game logic is concerned, is limited to a position on a tile grid. This makes collision detection easy, but limits the control over the character's movement. Because of these limitations, pure tile-based method is not suited for platform games that emphasize action or precision.

The levels in tile-based games are composed of multiple tiles that form a grid. Each individual tile on the grid includes information about what sprite it uses, what kind of sounds the make etc. Even the characters in pure tile-based games are composed of one or more tiles that are bound together. Though the movement of the character is limited to a grid in the logic, it can be made to look fluid with animations.

Moving the character with this method is simple. Whenever the character is about to move, a copy of the character is made on the adjacent tile according to the input direction. If the copy is overlapping a tile that is considered an obstacle, the movement is invalid, the character does not move and the copy is deleted. Otherwise, the movement is valid and the character is moved to where the copy is before deleting it. When it comes to jumping, the games usually limit the possibilities to strictly vertical or horizontal jumps. Diagonal movement in these types of games is rare, which is why the jump arcs are only visual effects rather than actual physics.

The advantage of pure tile-based method is that the game logic is very simple and it does not need as much fine-tuning as the other methods. Due to its simplicity, it

generates less errors and is easier to debug. Furthermore, it requires less effort to implement new mechanics to the game. (Monteiro 2012)

3.1.2 Tile-based (smooth)

Similarly to the pure tile-based method, the smooth tile-based method also features levels that are composed of a grid of tiles. Figure 8 displays the difference in these two methods, which is that in this one the character is not bound to the grid. The smooth tile-based method was the most common practice on 8- and 16-bit consoles and is still very popular today as it is very flexible, easy to implement and makes creating levels simple.

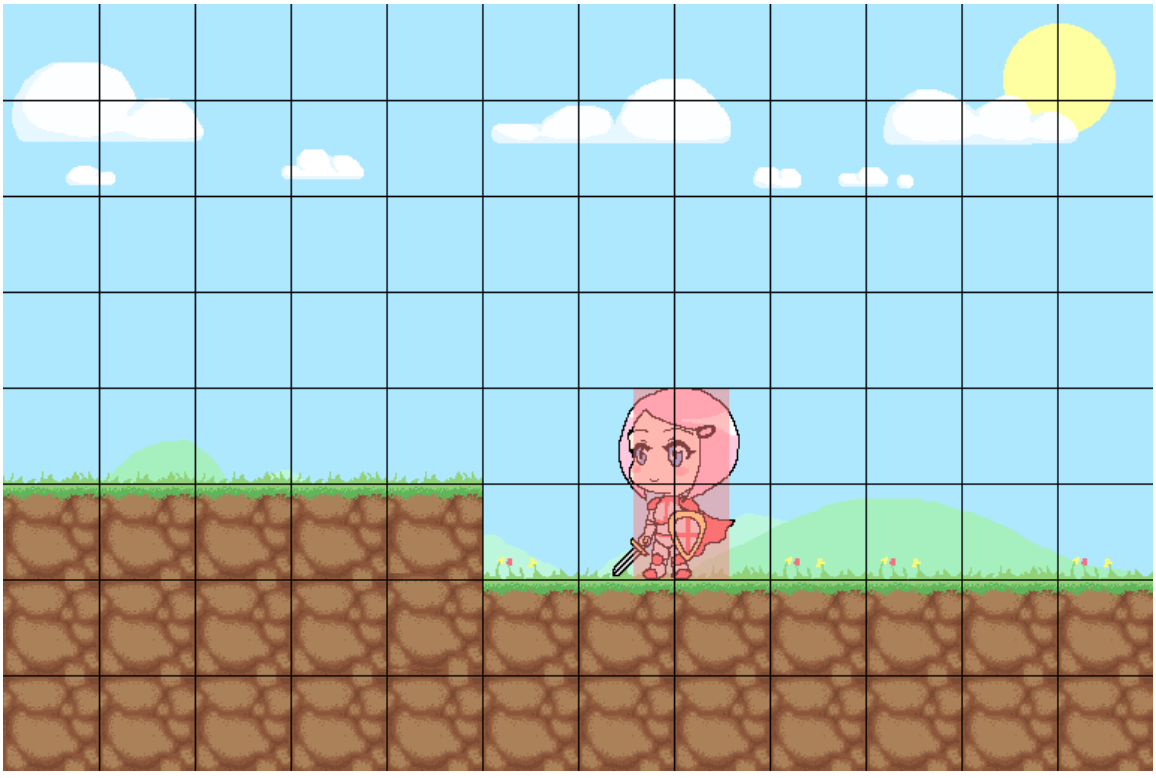

Figure 8. Character's position is not bound to a grid of tiles.

The smooth tile-based model uses the same kind of levels as the pure model, the distinction between pure and smooth being that in the smooth method, the character's position is not limited to a grid. To check collisions between characters and the level, each character has an axis-aligned bounding box (AABB) that is aligned with the x-, y- and z-axis of the world. The AABB is essentially a rectangle

that cannot be rotated and which outlines the character. The character's sprite/model can be larger than its AABB to make sure there is no collision when the character does not visually collide with the level or other characters.

The movement in pure tile-based model is handled individually for each axis. The velocity on an axis is applied as follows: first find out which rows of tiles the edge of the bounding box facing the movement direction is overlapping with. Then beginning from the tiles that the edge is overlapping with, those rows are looped along the direction of the velocity until an obstacle is found. After that, each moving platform is compared with the closest obstacle to see which is closer. Finally the distance between the edge of the character's bounding box and the closest edge of the closest object is calculated and compared with the distance the character is attempting to move. If the distance between the edges is shorter than the distance the character would move, the movement distance is clamped to the distance between the edges. (Monteiro 2012)

Slopes

Slopes can be rather challenging to implement, because they are considered obstacles if approached from the higher end of the slope, but the characters can enter the tile if they approach from the lower end of the slope. Even the lower end of the slope can be considered an obstacle if the slope is incomplete and the lower edge does not align with the tile grid. When adding slopes into a smooth tile-based game it is also important to make sure that the collision are checked on the x-axis before the y-axis, because movement on the x-axis causes movement on the y-axis if the character on a slope.

The tiles that form a slope each contain information about the height of its edges on y-axis for calculating the height between those two points, as indicated by Figure 9. So, when a character is standing on a slope tile, its position on the tile's x-axis is used to linearly interpolate between the heights of the tile's edges to determine the character's current position on the y-axis. While a character is moving up a slope, the collisions with the other tiles on the same horizontal row of

tiles as the slope tile should be ignored. This is to prevent the character from getting stuck at the end of the slope. (Monteiro 2012)

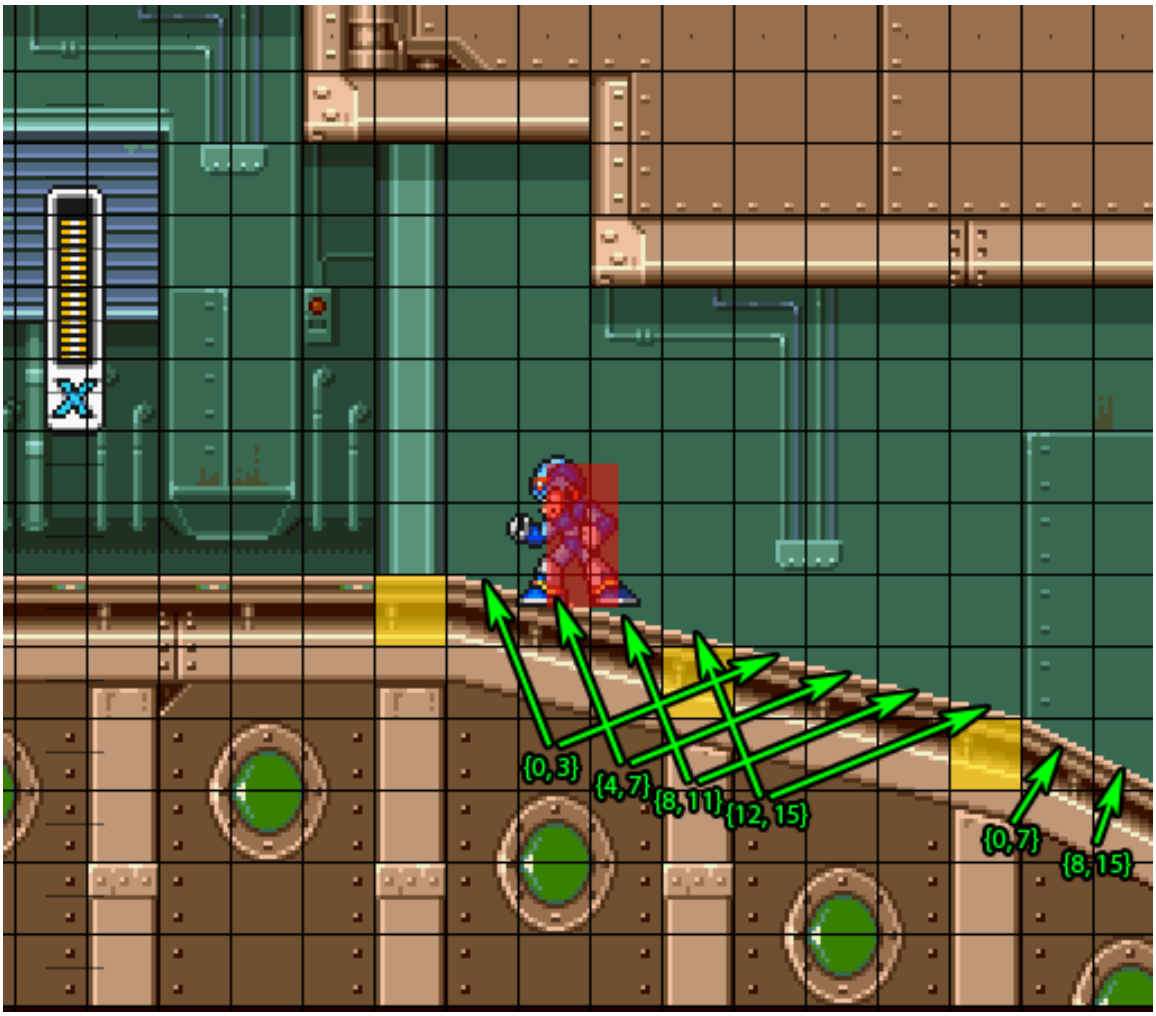

Figure 9. Slope from Mega Man X (1993), with slope tile annotations.

Ladders and Stairs

Ladders are generally one tile wide, and a character can start climbing them anytime the character's bounding box overlaps with a ladder tile by pressing up button. It is also possible to climb the ladder when a character is standing over a ladder tile and presses down button. When a character grabs the ladder its position on x-axis is aligned to the ladder, the gravity is ignored and the character can only move up and down the ladder.

There are multiple ways to exit a ladder in games. The most common ways to exit a ladder include; climb it all the way down, letting go of the ladder, climbing the

ladder all the way up whereupon an animation of the character rising up from the ladder is played, moving left or right if there are no obstacles and jumping of the ladder.

Stairs are a rare variation of ladders and work very similarly. The distinction is that on stairs a character moves on both x- and y-axis either one or half a tile at a time. In some games stairs are just a visually different slope. (Monteiro 2012)

One-way Platforms

One-way platforms are, as the name suggests, platforms that characters can stand on, but can jump through them for example from under the platform. These platforms are not considered obstacles on the x-axis and are only considered obstacles on the y-axis if a character is completely over the platform before the collision check. Some games have the possibility to drop through one-way platforms when certain conditions are met. This can be achieved by ignoring the collisions during one frame and making sure that the character is moving downwards, or by simply moving the character one pixel downwards so that it overlaps with the one-way platform. (Monteiro 2012)

Moving Platforms

Just like characters, the moving platforms also have an axis-aligned bounding box to check collisions with characters. Moving platforms should always be moved before the characters, because a moving platform can change the characters' positions. The game logic of moving platforms include checking whether there is a character standing on a moving platform. If so, its position relative to the platform is stored and then the platform can be moved. Finally the character who was standing on the platform is moved to the same relative position than it was before moving the platform. Once all the moving platforms have been updated, the characters can be moved normally. (Monteiro 2012)

3.1.3 Bitmask

Bitmask method is very similar to smooth tile-based method except the grid is pixel sized, meaning each pixel is considered an individual tile. This is also the reason why it uses more memory and is more complicated than the previous methods. Making levels when using the bitmask model also requires more work from the graphic artist, as the levels are made from full images instead of repeating tiles. This is also the reason why the bitmask method is rarely used on platformer, even though it can accomplish more than the other types. Bitmask also makes dynamic environment a possibility, which is the most common reason to use bitmask. (Monteiro 2012)

Slopes

Collision checks on bitmask method is the same as in smooth tile-based method, except when it comes to slopes. When checking for collisions on bitmask type platformers, the algorithm is almost the same as the one used in smooth tile-based type, except when checking the movement on x-axis, the character is moved few pixels up before searching the closest obstacle. This is to allow characters to move up slopes. Every time a character is moved on one axis, if even one pixel overlaps with an obstacle, the movement is reverted.

Because in this technique there is no distinction between falling and walking down a slope, the game needs to keep track on how many frames the character has been midair, so it can figure out whether the character is able to jump. It is also important to take in consideration how many pixels a character can move at a time without colliding with an obstacle. Bitmask type of platformers require a lot of adjustments to be made, and is less reliable than the tile-based methods. (Monteiro 2012)

3.1.4 Vectorial

Vectorial platformers use lines and polygons to check collision and is very hard to implement well. Despite this, its popularity is on the rise because of the availability of commercial physics engines. Vectorial method has similar pros as the bitmask without using as much memory.

This method also has it downs as it can sometimes be hard to tell whether a character is colliding with a wall, standing on a platform or sliding down a steep slope. Friction can be another problem as it is usually desired to be high on the feet, but low everywhere else.

One way to solve these problems is to give the character multiple bounding boxes with different properties. For example one box can be the feet, one head, two can be the sides and one can be the torso. Checking collisions for certain box gives information about the current status of the character. (Monteiro 2012)

3.2 Movement

The movement in platform games usually includes at least two features; running and jumping. Physics such as gravity, air resistance, etc. also play an important role in platform games, as movement is a core mechanic of any platform game. In platformers, the player's input often takes priority over everything. For example if the character is in the middle of an attack animation, and the player pushes the jump button, the animation is cut off and the jump animation is played as the character jumps. A good rule of thumb is that the game should always respond to player's expectations. (Pignole 2014)

3.2.1 Acceleration

In platform games, the acceleration is often divided into horizontal and vertical acceleration. If the character has horizontal acceleration, it takes some time for

the character to reach maximum running velocity, as demonstrated by figure 10. It might also take some time for the character to stop. However, it is recommended that the acceleration is implemented so, that the character can stop quickly, if not instantaneously. Otherwise the controls might feel slippery. Some platformers do not have horizontal acceleration, so the character always moves at full speed and is able to stop immediately. The horizontal movement can also take an analog controller into consideration, by having a target velocity variable that is adjusted according to the input from the analog.

Figure 10. The difference of having horizontal acceleration.

Even if the game does not have horizontal acceleration, it should have vertical acceleration. If a platform game does not have vertical acceleration, the jump arcs will be triangle shaped and the jumps will not look natural. With vertical acceleration, the jumps look more pleasant, and are easier to control. (Pignole 2014)

To define the characters target velocity variable, the input must be checked first. If the character is moving to positive direction horizontally, the target velocity should be the positive maximum velocity the character is able to gain. In the opposite case, the target velocity should be negative maximum velocity. If the character is not moving, the target velocity should be zero. Finally, if the player is in mid-air, the target velocity on Y-axis should be the maximum falling velocity due to the fact, that the character is constantly affected by gravity.

After the target velocity has been defined, it can be used to calculate a new velocity for the character. One way to calculate the new velocity is to get weighted average of the acceleration using a value between zero and one. This value is then used to linearly interpolate between the target and current velocity. Another method includes adding the acceleration to the current velocity, and making sure it is not greater than the maximum velocity. (Monteiro 2012)

3.2.2 Jump

Implementing jumping can be as simple as adding an upwards impulse to the character, and let the gravity (which is a common feature in commercial game engines) handle the rest. When the player jumps, there are four things he/she can affect; the impulse, horizontal movement, height of the jump and possibility to jump again in mid-air.

The impulse is usually a vector variable that is added to the character's velocity one time once the input is detected. This causes exploding movement in the character, which is perfect for implementing jumps. When the character jumps, the current velocity should be taken into consideration, so that horizontal movement does not stop when jumping.

A common feature in platform games is air-control. This means that the player is able to move the character horizontally in mid-air, even though it is not realistic. This gives the player more control over the jump arc and a possibility to do very precise jumps. However, it is important to remember that too much control and not enough control can make it hard to land a jump accurately. (Monteiro 2012)

As shown by figure 11, there are two ways to implement a jump; analog and non-analog. The non-analog jump means, that every time when the character jumps, the same impulse is added to the character's velocity. In other words, the height of the jump is always the same. The analog jump gives the player possibility to affect the height of the jump. When the player presses the jump button down, impulse is added to the character and a timer is activated. If the timer is active and the player is holding down the jump button, more impulses are added to the character making it jump higher (Pignole 2014)

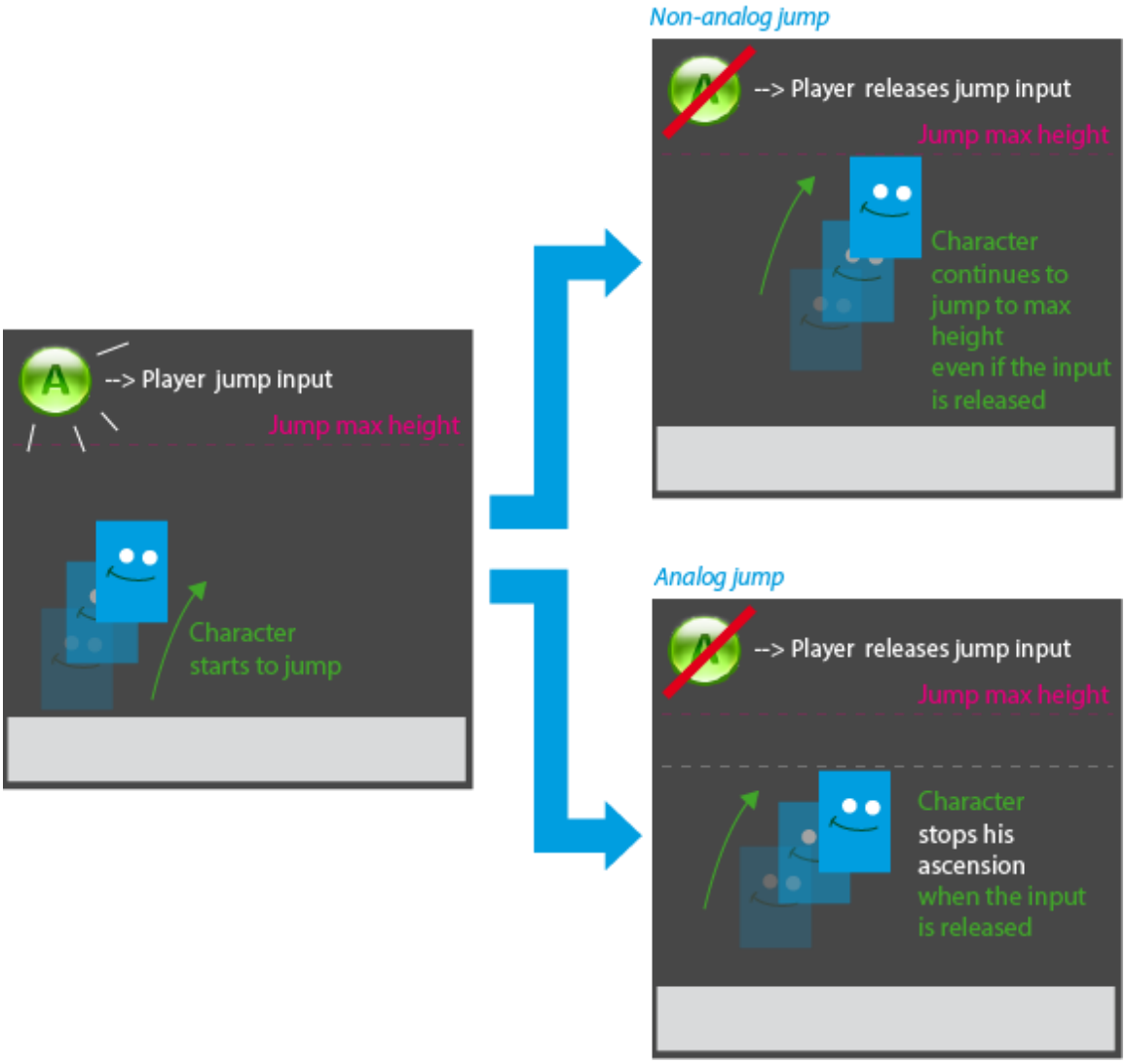

Figure 11. The difference between analog and non-analog jump.

Double jump is a common feature in platform games. Basically having a double jump means that the character is able to jump one time in mid-air. This can easily be implemented by adding a counter variable to the character, which counts how

many times the character has jumped without landing. By using the counter, the times that character is able to jump can be limited to a desired number. Once the character touches the ground, the counter is reset and the character is able to jump again. (Monteiro 2012)

3.2.3 Considerations

When creating a platform game, the following considerations are recommended to enhance the player experience. First of all, the human brain tends to predict a bit. Therefore, the player should be able to jump again just before landing. This can be achieved by giving the character a small tolerance. If the character is less above the ground than the tolerance value is, the player is allowed to jump.

Another consideration is related to falling of the edges. It is an undesired outcome for the player to just run off the edge, while thinking he/she would have been able to jump. To avoid this, the character can have a timer that activates when the character is no longer standing on the ground, but has not jumped yet. When the timer is active, player is allowed to jump though the character is not touching the ground. (Pignole 2014)

3.3 Camera

Despite often going unnoticed, camera is an essential part of any game. A well implemented camera allows the player to focus on the important aspects of the game without even realizing it. On the other hand, a poorly implemented camera can ruin the whole experience. A good rule of thumb when working on a camera is to have as little movement as possible while having the focus on what is important to show. (IMakeGames 2012)

Cameras are usually more like a collection of different features rather than a textbook execution of a single technique. They should be tailor-made for each game and should consider the following points; attention (what the player needs to see), interaction (how the player expects the camera to react) and comfort

(making sure that player is aware of the surroundings and does not receive conflicting sensory inputs). (Keren 2015)

3.3.1 Movement Conditions

The simplest way to implement a camera is to have it stay still at a fixed position. This gets rid of multiple problems that the other models have. However, not moving the camera affects the design radically as they need to be design with the fixed viewpoint in mind. Another problem emerges when targeting different screen sizes and resolutions as the players can see more or less of the level depending on their screen and resolution. (IMakeGames 2012)

The other extreme is having the camera locked to an object (usually the main character). This way the center of the view is always exactly on the target following it as it moves. The problem with this is, that it makes the camera move and stop quickly and makes the target seem still. Smoothing the camera movement usually helps to make it more comfortable. (Keren 2015)

A way to combine the previous method is what the author at the IMakeGames calls windowing, which is presented on figure 12. This term means having an area on the screen where the target can move without the camera moving at all. But when the target is about to surpass this threshold, the camera starts to follow the target. You can think of this as having a window that the target drags along as it moves. (IMakeGames 2012)

Figure 12. The camera's 'window' area represented in Rastan (1987).

Another technique that some games use is snapping the camera into a position when certain conditions are met. For example moving the camera to the main character's y-coordinate whenever it lands or moves on a platform. This removes the need to constantly move the camera up and down as the player jump making it easier to land precise jumps. (Keren 2015)

The camera can also be made to follow a certain path as the target move onwards, making it easy to manage what the player sees as he/she progresses through the level. This technique works especially well with 3D games that have two-dimensional gameplay. (IMakeGames 2012)

Some control over the camera's view can also be given to the player. It can be the ability to spin the camera around a 3D character, the capability to see further vertically to ensure it is safe to jump down or anything else the game needs. When giving control to the player, it is important to remember that the controls should be as intuitive as possible. (Keren 2015)

3.3.2 Smoothing

Smoothing camera movement is important and used in most platformers. It keeps the players from getting disoriented and makes the gameplay fluid. There are two popular ways to do this; lerp- and physics-smoothing. Lerp-smoothing as the name suggests uses linear interpolation to movement velocity of the camera is directly proportional to the distance between the target and the center of the camera's view. So, the closer the camera is to the target, the slower it moves and vice versa. One thing to consider when using smoothing is that the farther behind the camera lags, the less the player sees forwards. (IMakeGames 2012)

In the physics-smoothing, the camera is a physics object that constantly applies force to itself to move towards the target. This means that the camera can overrun the target point resulting in organic and fluid camera movement. However, because the camera's movement is slow to react to swift changes in direction, it can pose problems with fast moving targets. (Keren 2015)

3.3.3 Focus

What is even more important than how the camera moves, is what the camera shows. The most common focus being the center of the main character. However, in many cases it is beneficial to have the focus somewhere else. Figure 13 demonstrates the difficulty of deciding what to show to the player.

Figure 13. Figuring out what to show can be hard.

One fairly frequent model of camera focus is the forward-focus, where the camera is shifter slightly forward to the direction of level progression. This way the player can see what is ahead and it also serves as a motivation for the player to move towards the center of the view. If the level isn't linear or requires the player to go back, the forward-focus can be made to react to the direction the main character is facing or even react to the player inputs. (Keren 2015)

For more level specific needs on camera focus, anchors, attractors and repulsors are the solution. The anchors are used to set the camera to the anchor's position when the target enters a specific region of the level. This can be useful for example during a boss fight where the player needs to be constantly aware of the current situation and relative positions. The other level cues; attractors and repulsors do exactly what their names suggest. The attractors draw the camera towards themselves if the target is close enough and the repulsors push the camera away. These level hints are a neat way to have the focus on what is important and what is not, for example an attractor could be placed on an item to emphasize how important it is. (IMakeGames 2012, Keren 2015)

When dealing with multiple targets, it is usual to use the average position of those targets as the focal-point. The problem with this is that when the targets are far

apart from each other they might leave the view. A few ways to solve this is to restrict the targets movement so they simply cannot leave the viewport. The other way is to zoom out as the targets are close to the edges of the viewport. (Keren 2015)

4 HOW TO MAKE A GOOD PLATFORM GAME

4.1 Development

When starting a game project, there should be a clear vision, be it an epic story or just a small sentence. A clear direction makes a game consistent and robust as all aspects of the game take that story into consideration.

Defining a scope at the beginning of the development is a useful way to estimate the needed resources and time. It is too easy to fall victim to what is called the ‘feature creep’ where new features just keep piling up. Making it clear what features are essential and planned, helps the team to develop the game with those features in mind and to avoid unpleasant surprises.

A good way to get started with a project is to prototype. Prototyping is an excellent way to test different features and confirm they are actually fun. Trying things out with a fast prototype can make the game better and save a lot of time during the development of the final project. (Jonkers 2011)

In development, being efficient is often a good thing, because it makes development process faster. It does not mean cutting corners, but working smart. For example there is no need to avoid using commercial game engines like Unity or Unreal Engine. If the game does not require something special, there is no need to reinvent the wheel. (Hurst et al. 2015)

4.2 Design

Level design is obviously a major factor in making a good platformer. Good levels guide the player and make good use of the gameplay mechanics. The levels should promote variety in gameplay and aesthetics to keep the player engaged. (Boutros 2006, Koncewicz 2009)

Rewards are a good way to motivate players to test their limits, to explore and to play again. Many successful platformers also have collectibles that can be used to guide the player or to tempt them. Out of different types of collectibles the power-ups seem most effective resulting in immediate player enjoyment.

Surprisingly, the most acclaimed platform games choose to use as few buttons as possible. Often in these games the buttons could be used for multiple purposes. The median average of buttons used in these games is three. (Boutros 2006)

4.3 Graphics

When it comes to graphics, clarity is essential. The players need to be able to tell the foreground from the background and identify enemies, items, collectables etc. with a single glance, see figure 14. If the information is not clearly visible to the player, it can make the game really frustrating to play. (Jonkers 2011)

Figure 14. Distinct background and foreground from Limbo (2010).

Besides conveying information, graphics should be appealing to the player. The visuals play a key part in promoting the game and make the game more relatable. The graphics can also make the game more memorable for example Boutros

found that a big head-to-body ratio is more likely to be remembered. (Boutros 2006, Hurst et al. 2005)

4.4 Gameplay

As discussed before, responsiveness is extremely vital in platform games. Accurate and immediate feedback to inputs lets the player feel in full control over the character. The inputs should be intuitive and easy to learn so the player can concentrate on the fun. The challenge should not be learning how to play, but how to master the game. A sense of mastery and progression is what makes games worthwhile and it is one of the reason why games are replayed. (Koncewicz 2009)

5 CREATING A PLATFORM GAME

5.1 The Idea

I love 2D platform games. One of the reasons for this might be that the very first games I ever played include Super Mario Bros, Sonic the Hedgehog and a few games of the Megaman-series. So when I was thinking about a good subject for this thesis, I finally decided on platform games as I probably want to make platform games in the future.

The original idea was to start developing a game that was initially created during a game jam. Figure 15 shows the original game, which was a puzzle platform game where the core mechanic was to switch between side view and top-down view. I had been developing the idea further in my head for a while, but in the end I realized it had become too immense for a single person to accomplish in a reasonable time.

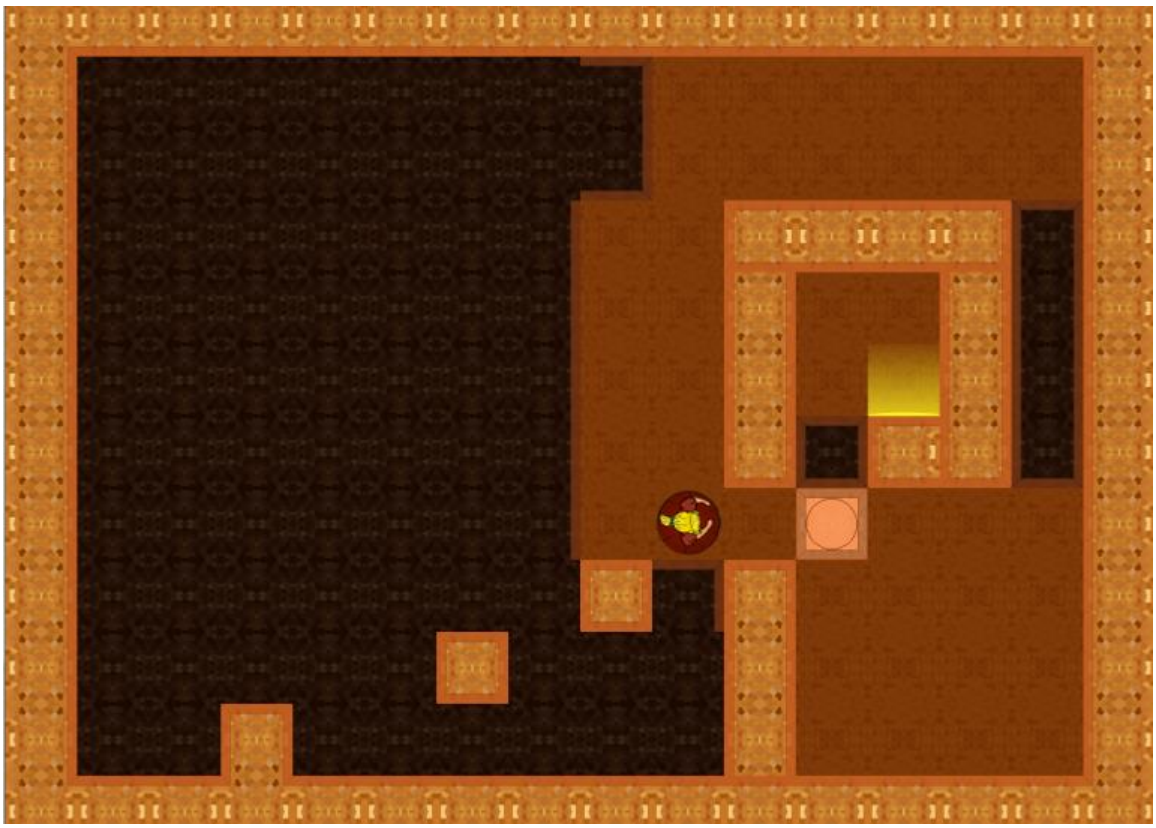

Figure 15. Screenshot of the game from Finnish Game Jam 2014.

I gave up on the puzzle platformer idea and decided to make an extremely simple game that would fit the scope of this thesis. The idea I began to work on was a one touch platform game for mobile. It would utilize a lot of the theory covered in this thesis and seemed viable to accomplish. The game would feature a main character that is constantly moving forwards and by tapping the screen the character would jump. The character would have a double jump to make the game a bit more diverse and the game would also have simple hazards that result in game over when touched.

5.2 Development

For the tools, I chose to use Unity (Unity3D. 2016), because using an off-the-shelf game engine greatly speeds up the process and out of the well-known game engines, such as Unreal Engine and GameMaker, Unity is the one I'm most familiar with. For the graphics, I used a program called Krita which is a free open source painting application. There are other similar and more popular applications like Photoshop and Paint Tool SAI, but they do cost some money unlike Krita.

5.2.1 Physics

Despite the fact that Unity has support for collisions that use polygons, I chose to use the smooth tile-based model instead of the vectorial one. I did this because: Firstly I would learn more by using the smooth tile-based method and implementing it by myself. And secondly, it would make creating levels a lot easier. So instead of using the Rigidbody component that Unity provides, I made my own script that had the AddForce and AddImpulse methods.

Having my own script for physics allowed me to implement collision checks inside the same script that handles movement. Whenever force is added, it is converted into velocity and added to a vector variable simply called velocity. On FixedUpdate the script also adds the velocity caused by gravity to the variable and adds the current velocity to another vector named offset. The script checks collisions for x-

axis and y-axis separately limiting the offset value to prevent the character from overlapping with obstacles. If the offset value on the axis was limited, meaning that the character collided, the velocity on that axis is set to zero. The collision check also sees if the character is colliding with a hazard obstacle resulting in game over and examines if the character is on the ground. Once the collisions have been checked, the character is finally moved by adding the offset variable to the current position and setting the offset to zero vector.

For the movement, I'm using the AddForce function. Simply adding force to the right makes the main character move. As a side effect for using force to move the character, it also has horizontal acceleration. Since I did not implement any drag, I just added an arbitrary limit to the horizontal velocity to prevent the character from accelerating indefinitely.

Character script calls the AddImpulse function from the physics script to make character jump any time the screen is tapped and the character is able to jump. The character script has an integer variable that count how many times the character has jumped. The counter is reset to zero whenever the character is touching the ground and it is used to limit the times the character can jump without touching the ground. When jumping the velocity on the y-axis is set to zero if it is below zero, meaning the character is falling. This is to make the jump consistent even in mid-air. The physics script takes into account a small threshold that is allowed between the character and the ground. If the character is not touching the ground but inside the threshold, it is considered to be on the ground. It also has a timer that is activated when the character is no longer touching the ground. If the timer has not reached a specified time, the character is allowed to jump even if it is not actually on the ground anymore. These considerations are made so the player would not feel like the game is 'cheating'.

5.2.2 Graphics

The graphics were definitely not a focus of this thesis and the game does reflect that with its simple and modest graphics. The game uses a single sprite for all of

its visuals differentiating different objects with color, as can be seen on figure 16. The sprite used is a square with cut corners. The reason for the corners is that it makes it a bit more interesting and when the sprite is used as blocks for the level, the indentations that are formed serve as landmarks for the player promoting learning. I also set the camera to perspective mode and added multiple sprites on different positions on the z-axis to create a parallax effect making the background livelier.

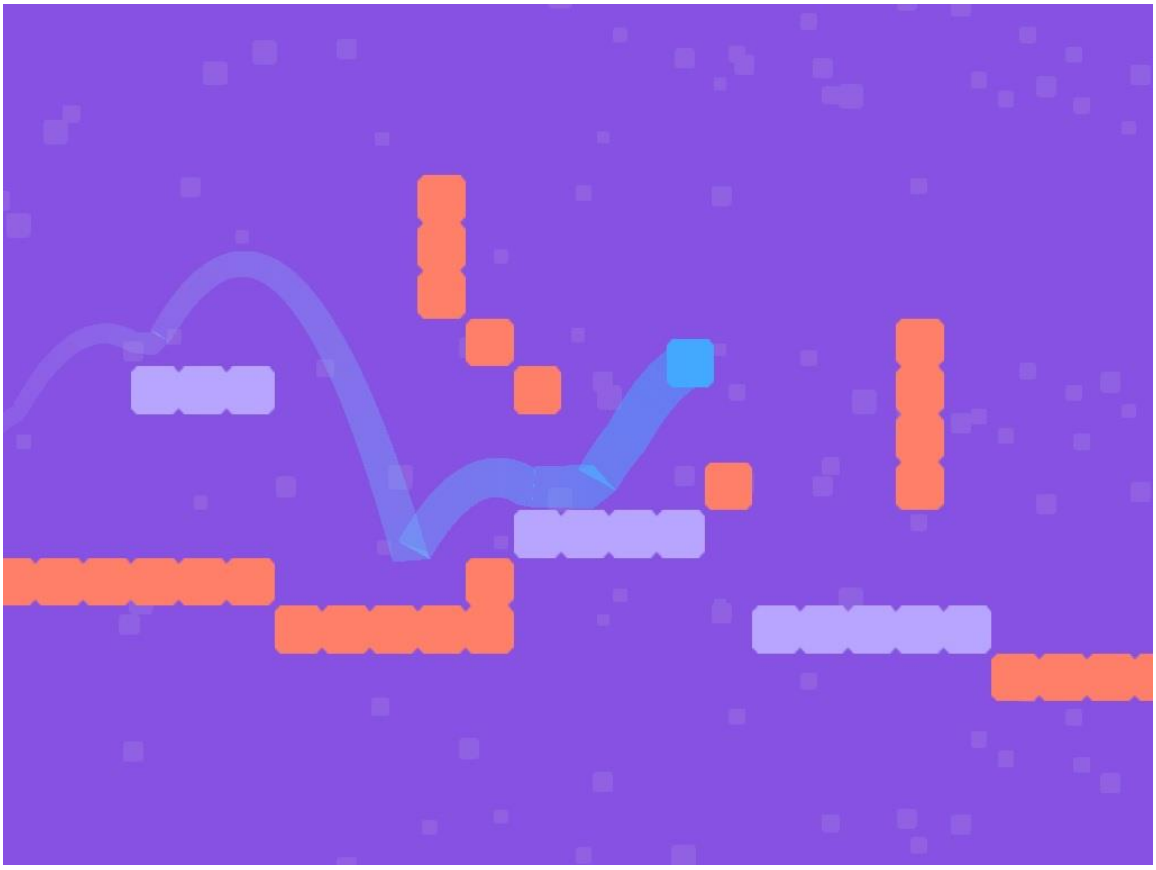

Figure 16. Screenshot of the thesis project.

5.2.3 Level Design

Even when designing levels for a simple game like this, there are still multiple variables that can be used when creating the levels. Firstly, the player is allowed to jump one or two times per obstacle. The levels can force the player to jump only once or to jump twice to clear it. Player can also be forced not to use the ‘first jump’, fall of the platform and then jump. Secondly, there is the timing. The player

needs to time both of the jumps accurately to pass. There can be many variations of which jump or jumps need to be timed perfectly. Thirdly, the two jumps may be timed however the player wants, but the player needs to land precisely. Finally, the levels consist many obstacles, so their frequency is a factor as well as the length of the level.

I made three levels that get harder the higher the number is. The length of the levels gets longer and the obstacles more frequent. The obstacles on the last level need more precision than the ones on the previous levels. This helps the player to get into the game and get better as the game progresses.

5.2.4 Camera

For the camera, I decided to use simple generic features. I used position-locking and lerp-smoothing to create a camera that smoothly follows the main character. The smoothing helps to convey the movement of the character and makes it easier to look at and to be aware of the character's position relative to the world around it.

5.3 Analyzing the Game

5.3.1 Pros

The game is intuitive and easy to get into. It is surprisingly entertaining and challenging and makes you want to try again until you pass the level. It is quick to play and it is very effortless to try again. The mechanics work well and the game seems mostly fair. The graphics are easy to read and there is a clear distinction between what is part of the level and what is background.

5.3.2 Cons

The game is really bare bones; it does not have audio and is overall really minimalistic. The graphics are not bad but they are not exactly appealing either. The player can think of the collision being unfair, because the sprite I used is not a perfect square, but the collisions are calculated as if it was. The camera works as intended, but it could be more suitable for this particular game.

5.4 Lessons Learned

First of all, I come to realize that keeping it simple is good. Having great core gameplay that is polished is very valuable. Sometimes we just keep inventing cool new features without realizing they just make the game unpleasantly complex and hard to approach. If the core mechanic cannot stand on its own then there is not much that new features can do to improve it.

Clearly I learned to implement the smooth tile-based model which is a good versatile method that can be used to make games other than platformers as well. Making the character move and jump can sound easy, but it can actually be very hard to make it properly.

I also came to notice, that there are many little aspects and features that I was not aware of before. For example the considerations for allowing the player to jump when it would normally be impossible or how many different ways there are to implement a camera.

Reading the theory behind how cameras work in games helped me to realize the importance of what players see and how the camera movement affects them. I also learned how much variety there is in the different popular features cameras have in games and how they are tailor-made to fit the game.

6 CONCLUSION

Platform games have come a long way from the 1980s remaining ever popular throughout the years, evolving alongside the hardware. Even today, as virtual reality is becoming more common, there are people developing new platformers to face the challenges that come with it. This is made possible by having a firm grasp of the basics and an open mind to learn.

There are multiple different ways to create platform games, and all have their own merits. There should always be a reason when using a certain method and each aspect of the game should take into account how well it suits the idea and the vision of the game.

Working on the camera can sound boring, but it is an essential part of any game. The camera allows the player to focus on the fun and the important parts of the game without feeling any discomfort.

In platformers and games in general, it can be easy to implement a feature, but it is the attention to the smallest details that make each individual feature so great. And it is harmony and the consistency between these features that make a game good as a whole.

REFERENCES

Bexander, Cecilia. 2014. The Cinematic Platfomer Art Guide http://www.diva-portal.org/smash/get/diva2:692737/FULLTEXT01.pdf (Read 12.10.2016)

Boutros, Daniel. 2006. A Detailed cross-examination of yesterday and today's best-selling platform games http://www.gamasutra.com/view/feature/1851/a_detailed_crossexamination_of_.php (Read 13.10.2016)

Chong, Ben. 2015. Endless Runner Games: How to think and design (plus some history) http://www.gamasutra.com/blogs/BenChong/20150112/233958/Endless_Runner_Games_How_to_think_and_design_plus_some_history.php (Read 12.10.2016)

Cifaldi, Frank. 2012. Sad But True: We Can't Prove When Super Mario Bros. Came Out. http://www.gamasutra.com/view/feature/167392/sad_but_true_we_cant_prove_when_.php (Read 24.10.2016)

Electronic Games. 1983. The Player's Guide to Climbing Games. Electronic Games **1**(11): 49. https://archive.org/stream/Electronic_Games_Volume_01_Number_11_1983-01_Reese_Communications_US#page/n47/mode/2up (Read 21.10.2016)

Fash, Travis. 2008. The Leif Ericson Awards. http://www.ign.com/articles/2008/03/24/the-leif-ericson-awards?page=2 (Read 24.10.2016)

Flecther, JC. 2009. These Metroidvania games are neither Metroid nor Vania https://www.engadget.com/2009/08/20/these-metroidvania-games-are-neither-metroid-nor-vania/ (Read 13.10.2016)

Games Radar. 2010. Gaming's most important evolutions. http://www.gamesradar.com/gamings-most-important-evolutions/?page=3 (Read 24.10.2016)

GameZone. 2007. Sonic 3D Blast, Wrecking Crew, Super Air Zonk http://www.gamezone.com/originals/sonic-3d-blast-wrecking-crew-super-air-zonk (Read 26.10.2016)

Gorenfield, Louise. 2013. http://www.extentofthejam.com/pseudo/ (Read 26.10.2016)

Guinness World Records. 2008. First platform videogame in true 3D. http://www.guinnessworldrecords.com/world-records/first-platformer-in-true-3d (Read 26.10.2016)

Harrel, Wayne. 2013. The Indie Revolution: How little games are making big money. https://www.gameacademy.com/the-indie-revolution/ (Read 26.11.2016)

Hurst, Steven; Mayles, Steve; Price, Gavin; Restemeier, Jens; Sutherland, Chris; Stevenson, Mark. 2015. 6 Musts for a perfect platformer, from the Yooka-Laylee team http://www.gamasutra.com/view/news/243310/6_musts_for_a_perfect_platformer_from_the_YookaLaylee_team.php (Read 9.11.2016)

IMakeGames. 2012. Cameras in 2D platformers http://www.imake-games.com/cameras-in-2d-platformers/ (Read 8.11.2012)

Jonkers, Diorgo. 2011. 11 Tips for making a fun platformer http://devmag.org.za/2011/01/18/11-tips-for-making-a-fun-platformer/ (Read 9.11.2016)

Klappenbach, Michael. 2016. What is a Platformer http://compactiongames.about.com/od/gameindex/a/platformer_def.htm (Read 6.10.2016)

Keren, Itay. 2015. Scroll back: The theory and practice of cameras in side-scrollers http://www.gamasutra.com/blogs/ItayKeren/20150511/243083/Scroll_Back_The_Theory_and_Practice_of_Cameras_in_SideScrollers.php (Read 8.11.2016)

Koncewicz, Radek. 2009. What made those old 2D platformers so great? http://www.significant-bits.com/what-made-those-old-2d-platformers-so-great (Read 9.11.2016)

Monteiro, Rodrigo. 2012. The guide to implementing 2D platformers. http://higherorderfun.com/blog/2012/05/20/the-guide-to-implementing-2d-platformers/ (Read 20.10.2016)

Parish, Jeremy. 2014. Five Critical Moments in Platform Game History. http://www.usgamer.net/articles/five-critical-moments-in-platform-game-history (Read 24.10.2016)

Parish, Jeremy. Essential 50 http://www.1up.com/features/essential-50-mario-64 (Read 12.10.2015)

Pignole, Yoann. 2014. Platformer controls: how to avoid limpness and rigidity feeling. http://www.gamasutra.com/blogs/YoannPignole/20140103/207987/Platformer_controls_how_to_avoid_limpness_and_rigidity_feelings.php (Read 20.10.2016)

Racketboy. 2011. Platforming games 101: Running, jumping & more http://www.racketboy.com/retro/platformers/platforming-games-101-all-you-need-to-know (Read 13.10.1016)

Retro-Sanctuary. Run and gun games http://retro-sanctuary.com/Run%20and%20gun%20games.html (Read 20.10.2016)

Rybicki, Joe. 2008. Prince of Persia Retrospective https://web.archive.org/web/20080509013828/http://www.gametap.com/articles/gamefeatures/prince_of_persia_retrospective-05052008 (Read 12.10.2016)

Unity3D. 2016. Unity game engine. https://unity3d.com/ (Read 27.11.2016)

List of Figures