Extending a platform game in C/C++:

Michael Mbomena

Module Code: CS1PC20
Assignment report Title: Programming Project
Student Number: 29017995
Date (when the work completed): 23 March 2021
Actual hrs spent for the assignment: 20
Assignment evaluation (3 key points): engaging task, learned something, could improve the description about.

Introduction

Project goals

A game is generally made of at least three components: a goal, the obstacles and a feedback system. I don't want the game to be repetitive and along with that what a lot of games aim for is longevity and replayability, so I have aimed to focus my additions on that.

To this I want to provide players with multiple ways to complete a level to give them a sense of control and choice. Doing this can make them feel more immersed. For example, they could reach the end of the level using different paths. A feeling of decreased predictability can also be achieved by substantiating NPCs, health packs or ammos in random locations, with an added probability that the NPC may not act logically. NPCs could use random paths (this makes them less predictable). Unpredictable NPC behaviour adds to the challenge and creates a new experience instead of having the NPC's act in a predetermined pattern. As long as the game is essentially based on skills, adding random events should bring some interesting challenge.

I want it to be more in line with puzzle platformers such as Portal but as a 2D game. These have strong arcade roots, and known for being very difficult, and having very linear game progression.

In the game I want their to be the ability to change your jump in midair. Most platformers have falling from considerable heights or ditches within the game world that kill a character immediately were they to fall into them which is something I want in this game to have also.

I want to provided players with a sense of pride and accomplishment for achieving different goals and milestones and this can be done with rewarding players for taking risks such as a shortcut through the level which will complete the level faster but also has an increased risk of losing a life as there may be more complex platforms to navigate through.

Players play games for different reasons; it could be for exploration, escapism, competition, or just to challenge themselves. So, what is perceived as a reward may take different forms, based on the player's preferences. I want to provide a wide range of rewards in the game including score increase, extra lives, extra-powers, a spot in the high-score list, and new sound effects.

This means that the player can acquire new weapons, and skills such as a double jump. The idea is that the player's character will evolve overtime.

Game features

The main feature I have chosen to implement is a countdown timer therefore creating a time limit on each level. A time limit is almost always an obstacle or a feedback system and should always either remind you of your progression in the game or make it hard for the player to reach his goal. Along with increasing the actual puzzle difficulty, a time restriction could be another way to increase difficulty, and the combination of both can be used as a way to give a sense of progress. It makes the game more tense, enjoyable and instils a sense of urgency to the player making them feel more challenged. As it makes levels more challenging it also rewards practice, because you are better able to conduct a good strategy and actually play it out in the time limit as well as providing replay ability by trying to get faster times. They add meaningful complexity to a player's decision making and are only generally bad when they feel like a non-meaningful obstacle that only affects poor players. More challenging time limits will keep the player interested as their skill level in the game increases. Along with a timer I aim to develop incentives for completing levels at various rates encouraging them to be replayed, such as trophies and exclusively locked content as well as incrementally increasing the time limit when parts of the game are completed well. I believe these will be an interesting addition to the game. However, there will be an option for players to disable the timer if they just wish to explore the game or perhaps an option to enable a tired system where the level is timed but gives no bearing to whether you able to continue to next level or not instead giving you a letter grade from 'A' TO 'F', as those who don't care will still be able to progress onwards. It is also easy to test the acceptable time-limit needed for each level by just playing through it.

Another feature I have chosen to implement is an online high-score table. There is a similar feature in EA Sports' FIFA series. During the minigames before a match there are skills drills the player has an option to complete and when completed the player is able to see how their achieved score compares with their friends, and as expected this leads to a lot of competition. A progression bar at the top of the screen that shows how far you are along in the level is another feature that I'm thinking of adding as it helps the player gauge their current progress. A power up that can be unlocked when the player acquires enough points such as a double jump will help make the gameplay more dynamic, less static and more refreshing leading to longevity of the game. Game performance metrics are the last feature that shows the stats of the people during the level such as the number of jumps they did. I think this is just an interesting feature that can lead to replayability and competition by players wanting to complete levels in fewer jumps.

Justifying my choice of language and programming style

The main reason I choose to use C for the addition of new features is because it is the programming language, I am most familiar with. C as a language works best when you're

breaking down a problem into smaller subproblems as C is a much smaller, simpler language than C++, and the additions I have in mind don't have to be overcomplicated. It's easier to know it well. Also, the compile times for C++ code can be very long and annoying and in comparison, C is very quick. The standard is also shorter, easier to read and comprehend. C is easier to debug which can lead to quicker development of the features. There is no ned for C++ for me as the features I am focusing on adding like a countdown timer don't really need object-oriented programming to be implemented.

Design

Feature design

The timer is in the top right of the screen and counts down giving a time limit to how quickly you can complete the level. Of course, it is possible to fail because of running out of time, but this typically only happens when the puzzle is obtuse, or if the player is just stumped. It's mainly there to provide a sense of urgency. In some cases, the timer will be able to increase the amount of time it has left by the player competing certain puzzles with great efficiency. Effective use of timers is just another tool that allows for the manipulation of the player's mental state. The timer will be in the top right corner so it will be out of the player's main gaze and does not clash will any other UI elements and with a simple font and solid, single colour scheme that distinguishes it from the background so that the player does not have to focus too hard to gauge how much time is left. It will always be there because players would be dissatisfied and frustrated not always knowing how much time is there and also causes a sense of urgency.

The camera view of the game is designed so that the sprite character that the player controls is always in the centre and tracks the sprite horizontally only.

Highscore table – When you finish the level the highscore table comes up on screen and shows you where your scores ranks when compared with other people who have also completed the level.

The progression bar appears on top of the screen in centre and gradually fills the closer you get to the end of the level. It will be easily distinguishable from the in-game world

The double jump power up will be available as a power up. This can be used to return to previous levels where perhaps a particular route was out of reach as the character could jump that high at the time. This provides replayability for the players

Design description and gameplay functions

The player controls a character in which the primary movement is running and jumping onto and around platforms while the camera is centred on the character and horizontally scrolls across the screen. The aim is for the player to navigate their way across the map completing complex platform puzzles almost like a maze within the time limit while also while simultaneously collecting pizzas for points, doing this all while keeping within the time limit for the level. There are multiple levels and navigating each level becomes progressively harder as the difficulty in platform puzzles increases as well as the time limit changing and you gaining new abilities like a double jump the further you progress if you collect enough points The will also be multiple paths to navigate the game will the more challenging resulting in a higher reward. Previous puzzle sequences will show up later in conjunction with other puzzles in order to make sure that previous completions and the knowledge of them are validated with the player. There are no other entitles in the game besides the main character.

How the user interacts with the game

The player starts the game on the start screen. The player will be presented with a tutorial level that shows them the basic controls of how to play the game. The player moves the character with the WASD keys on the keyboard. The game communicates with the player what to do by presenting prompts on screen that tell the player what to do. The player can choose from multiple paths to choose from to explore. The narrative story is communicated to the player through the characters' speech and through onscreen speech bubbles that give current context of the situation. The narrative story is that the player has found themselves in a foreign land and has to navigate their way through it in order to get back home. The way some words are displayed will be changed through font size, and animation to make them more impactful to the player. This allows the player to feel a certain 'weight' to the game depending on the situation.

A game hurts itself by providing too little information or too much, requiring too many inputs, confusing the player with unhelpful prompts or making it hard for a new player to interact. The over-supply of obvious information makes it feel like the game doesn't trust the player to be competent. Poor UI design can even break the game completely. I want to provide information to the player clearly and easily without creating too much of a distraction. I want to keep it simple while also being visually appealing. I want the art style to be appropriate for the game, so it'll be minimalist with a simple colour scheme. For a game like this, a 2D platformer, I feel like less is more. The player will be presented with information and stats about their gameplay through non-diegetic UI components such as a health meter, point counter, and timer that shows how much time they have left. These will not take up a lot of space on screen and will instead be in the corner as it avoids over cluttering and enhances visibility of the actual game. The health meter will be in the top left, the timer in the top right and the point counter in the bottom right.

The player will receive visual and audio feedback such as sparkles or a ‘jingle’ when a challenge is completed well. This will make an event and action more enjoyable when finished and each effect makes you feel more engrossed in the game and makes it less boring

How the game play is supposed to progress

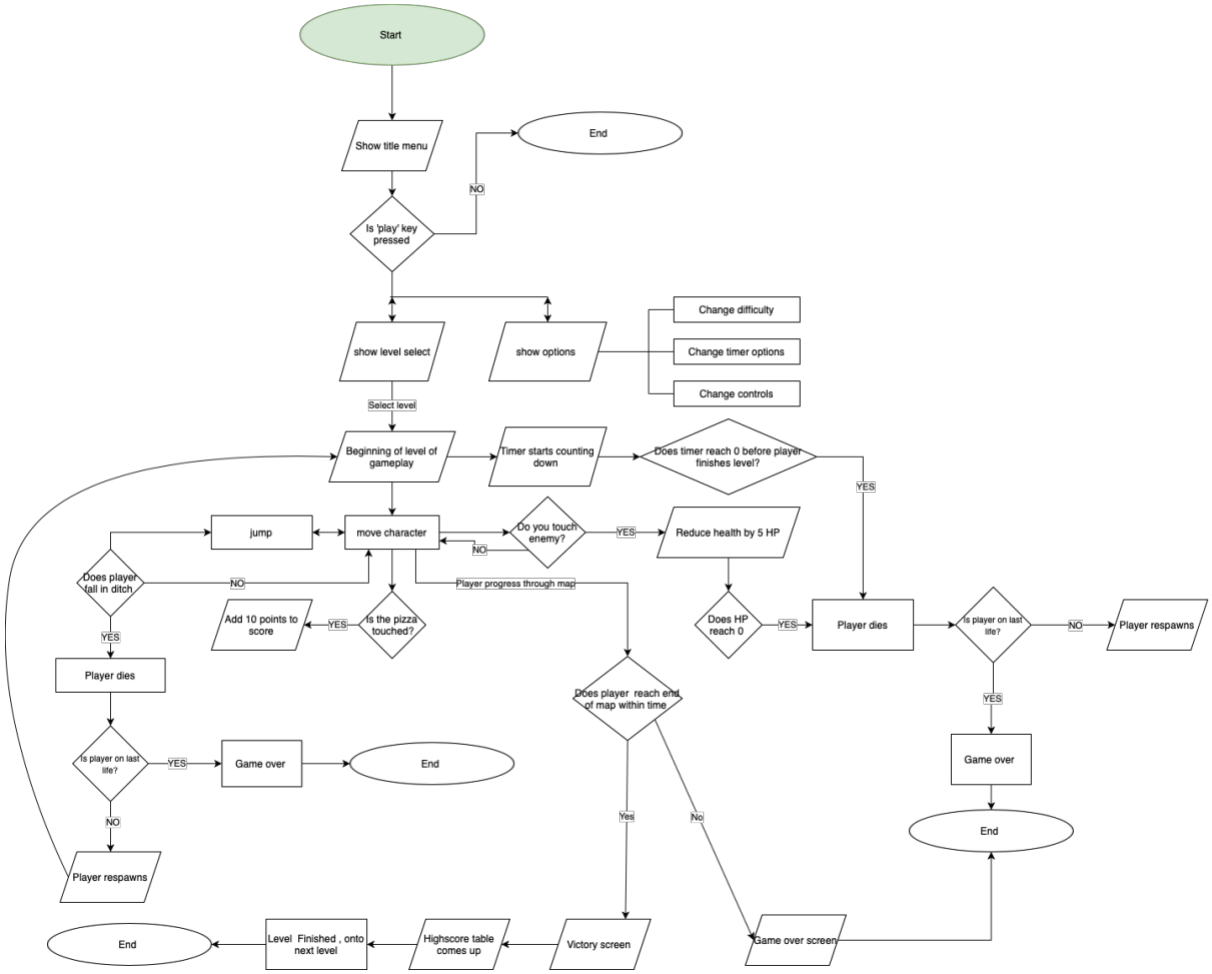
The gameplay progresses by the player collecting pizzas which adds to the points tally. The pizzas are in hard positions to reach making collecting them challenging. The more points the player collects it allows them access the different enhanced abilities such as a double jump or increased speed. This change in the game’s dynamics over time to keep the game interesting and difficult as the player increases in skill to reward continued play. The player has to complete the current platform puzzle they are on within the time limit before they can move on and they get progressively harder such as falls causing more damage which will call for a better execution of the player’s timing. As the game goes on the visual and audio rewards become more fanciful in nature. There will even be rewards that enable to change the appearance of your character.

If the distribution of the game’s progressive elements are not structured well there is a risk of overwhelming the player with too much upfront or not keeping the player engaged with the game because it is stale and seemingly going nowhere. The difficulty increase should be linear with the characters ability increase in order not to frustrate the player. Games like Nintendo’s ‘Zelda’ are carefully planned in this way creating a compelling experience that keeps players coming back. The progression should be structured in a way that feels fresh. By the end game the player should be able to easily complete past levels that at a previous point in time seemed challenging.

Changes I made to the initial design during the implementation phase

I didn’t really make any changes to the initial design.

Flowchart of playing of a game



Implementation and Development

How I developed the code and what language features you employed and why

C is a simple language in the sense that it provides an easy way to break the problem into smaller parts I used its set of library functions and data types. I used the pointer functions in C to create arrays. In the finished game when the level starts the timer starts a countdown.

Use of novel features

Description of changes from original design

My program stayed relatively consistent with the original design.

Development process

Testing

I found bugs in the game through Exploratory testing. Once something was implemented, I proceeded with playtests in order to iterate and refine quickly until I found the best way to improve it. I did not find all the bugs within it though. I also did functionality testing to look for issue with stability within the UI. My strategy for debugging was just to do it by brute

force by going through a lot of the code at once but this lead to a lot of wasted time and effort.

Conclusions

Reflections

One thing I remember is how hard it was to take the very first steps. Getting a development rhythm going really seemed unfathomable to me at first. When you try something and nothing you do directly translates to what you really want to build can be really discouraging. But I learned that the most important thing was to just start and get that positive feedback loop going.

It is important to split all your goals into tiny steps and focus on just the next immediate step. I was constantly realising my lack of knowledge and noticed the inefficient way I handled challenges presented to me and whenever I hit a serious wall, I realised it was okay to readjust, shift my focus and take a different path. I looked up little tutorials for small tasks that I got stuck on or read through stackoverflow, and just worked on them the best I could, so I could try to take that new knowledge and try to turn it into something functional.

Once I started becoming acquainted with some of the limitations of extending code for a game, a few of the fundamentals started looking like basic building blocks.

I thought there must be a better way than writing a new line of code for each one. This enabled me to focus on loops and arrays strengthening my proficiency with them. I learned how to use a data structure like I didn't know that array can be used to create platform games where the screen is just a 2D array, and different objects are just different integer values.

A big advantage I have not noticed is learning through visual feedback as you see the result of the piece code you have written, and it is clearer to you what has to be changed.

Another thing I learned is that you can't be overly ambitious as many features that you want to implement require an understanding of other concepts that you might know yet, so it's best to start small then slowly build yourself up otherwise you just end up trapping yourself. It's easier to get ahead of yourself and try to implement things that seem really cool but are just way out of your ability, I had big ideas and wanted to do things that take big team years to do.

Throughout I realised that it is okay to ask for help from other people on the course and this enabled me to understand harder concepts.

I learned how to use a data structure like I didn't know that array can be used to create platform games where the screen is just a 2D array, and different objects are just different integer values. I learned how to create sprites and sound effects.

What I've learnt about developing a significant program

Through developing a significant program, I learnt that if you're uninterested in a task, the amount of effort required to keep doing it gets exponentially bigger. On the other hand, being interested makes the task feel effortless.

It is also easier to start simple then gradually add one rule at a time instead of trying to design everything in one big effort. You have to make sure the foundation works first before trying to add anything complex. When trying to build on a buggy program the initial problems become hidden and harder to find and difficult to tell whether there's an issue with recently added components or something else. If you only add one new rule and a critical game system becomes broken in playtesting, you know exactly where the problem is, because you only changed one thing. It's better to add a new feature, playtest it to see that it works as intended then decide an appropriate course of action after.

It is also a good idea to get a second opinion about some of the ideas that wish to implement who isn't as emotionally invested as you are because they will be able to give an unbiased opinion and tell you whether your idea is good or necessary.

There are things that you will learn only when you develop, and finish your game, things you can only learn through experience. Even then there are still things you simply cannot control.

If I had more time I would implement the feature to add your own music into the because I always thought that would be a cool feature to have in a game. I would also add enemies into to the game to make it more challenging and have them designed would a random levels of AI to make the gameplay more unpredictable.

If I was to do things differently I would probably give myself more time and room for error.