Contents

[Analysis Stage 3](#_Toc198033362)

[Computational Thinking 4](#_Toc198033363)

[Thinking Abstractly 5](#_Toc198033364)

[Thinking Ahead 6](#_Toc198033365)

[Thinking Procedurally 7](#_Toc198033366)

[Thinking Logically 8](#_Toc198033367)

[Thinking Concurrently 9](#_Toc198033368)

[Stakeholders 10](#_Toc198033369)

[Interview 1 11](#_Toc198033370)

[Interview 2 12](#_Toc198033371)

[Interview 3 13](#_Toc198033372)

[Existing Solutions 14](#_Toc198033373)

[Essential Features 15](#_Toc198033374)

[Limitations 16](#_Toc198033375)

[Software and Hardware Requirements 17](#_Toc198033376)

[Success Criteria 18](#_Toc198033377)

[Design Stage 20](#_Toc198033378)

[Decomposition 21](#_Toc198033379)

[Structure 22](#_Toc198033380)

[Algorithms / potential solns 23](#_Toc198033381)

[Variables, Data structures, Classes 24](#_Toc198033382)

[Test Data 25](#_Toc198033383)

[Extra 26](#_Toc198033384)

[Implementation Stage 27](#_Toc198033385)

[Stage 1 28](#_Toc198033386)

[Stage 2 29](#_Toc198033387)

[Stage 3 30](#_Toc198033388)

[Testing Stage 31](#_Toc198033389)

# Analysis Stage

## Computational Thinking

The problem is solvable using computational methods because of the fact it is a game. Since the game is a platformer type of game, it will require inputs (to recognise what the player is doing so it can be drawn onto the screen or responses by the game can be made, especially movement and actions). When a player plays the game, the game also has to be interactive with the player since otherwise, the player is just making inputs and getting nothing out of it, which wouldn’t really mean they are playing a game, this means the game also has to output things to the player and runs calculations to tell what the player is doing. These require algorithms to do and therefore allows the problem to be solved via computational methods. The fact that the game has a form of repetition to it (due to levels, timers, scoreboards etc) also means that algorithms need to be used repeatedly to solve this problem.

### Thinking Abstractly

### Thinking Ahead

This way of thinking allows me to decide and plan ahead what I will do. This is important since if I do everything concurrently, there wouldn’t have been much time to properly think out what the best ways of accomplishing my goals are. It provides proper organisation so that when I do get to designing, and creating code, everything will be well organised so that I know exactly what to do, when, and how. This simply makes things easier for me since if I have to both think and do at the same time, I am more likely to make errors. Some things I plan on doing or using are:

* Using python; pygame to write my game as it is good for producing these type of retro games, especially since mine is pretty much all 2 dimensional and pygame is good for drawing 2 dimensional things.
* I know what base controls (standard controls) will be used such as the keys ‘w’, ‘a’, ‘s’, and ‘d’, which are very commonly used as movement controls in modern video games. This can be changed via settings that will be made later. This is also more comfortable for players that have already played platformer games before.
* Some outputs by the game are like sound effects to indicate player’s actions that caused something, such as buttons that can be interacted with,

### Thinking Procedurally

### Thinking Logically

### Thinking Concurrently

By thinking concurrently, I can identify issues with the current solutions and fix them. This is important since I know I will have a scoreboard and timer (a simple scoreboard and the sun that rises and falls to indicate timer), therefore I already know that I need these two features to be updated at the same time since as the player is obtaining score, the timer is also moving.

## Stakeholders

The game’s graphic design will be very similar to the three games mentioned earlier; arcade and retro style. This art style is simplistic and useful for my game because it ensures that players are more focused on the core gameplay rather than how the game looks. This is important since the point of the game is to improve problem solving skills, if the player is too distracted by a beautiful graphic design, they won’t be fully concentrated on solving the problems that are presented in the game. By having the player’s focus set on the core gameplay itself, the game will be able to more accurately assess a player’s current set of logical reasoning skills, and improve them more efficiently. The game is also intended for computer and laptops, this is because PC keyboards allow for flexible controls. Furthermore, a keyboard allows for a better approach to improving problem solving because if I were to implement mobile controls for example, I would have to use icons or decals to indicate what each control does, and this doesn’t really help the player in developing their mind because they will easily be able to tell what each control does, whereas forcing the player onto a keyboard where they don’t know what each key does, allows them to think about how they can use the keys to beat the game. Considering these factors, the game will be most suitable for audiences in the age group of 7 years to 18 years. This is because since the game is going to be available for PC only, and children of these ages are in school where computers are used more often, they will be able to play. The majority of children in this age group also likely already know the alphabet, so the keys on a PC keyboard will be understandable making them easier to understand which key does what (which will be explained in the tutorial of the game). Also, since some text is present in the game, the audience should be capable of basic reading which this age group should be capable of.

### Interview 1

### Interview 2

### Interview 3

## Existing Solutions

One game that is already a solution to my problem is ‘Dead Cells’, it is a fighting-adventure game where the player traverses multiple levels each with a different kind of layout and obstacles, along with unique enemies in each level. At certain points of the game there are also bosses. The player must reach the end of each level (a door) before they can proceed to the next level. Each level also contains other points of interest such as item/collectible shops and secrets/easter eggs. The game is 2 dimensional and seen from the side of the player and involves the player collecting weapons and upgrades to defeat enemies easier. Some of the player’s base features are the ability to run, attack, jump, roll (a defensive option that provides something called ‘invincibility frames’ (i-frames) where while the i-frames are active the player takes no damage and “dodges” all active attacks landing on the player), sliding, and drop slamming (where the player can negate fall damage by accelerating while falling and slamming into the first horizontal platform they fall upon). There are also many obtainable features that the player can obtain and keep. Every time the player defeats an enemy, depending on the enemy, they will drop a reward for the player, usually this reward is a form of currency the player can use (cells, which are all lost on death by the player as well) but occasionally, the reward may range from collectibles to attribute upgrades to weapons. If the player dies, they lose almost all of their possessions and are reset to the very beginning of the game. The game also features a healing system where the player has a number of ‘healing flasks’ which they can drink to regenerate health points, and these flasks may be refilled in the intermission stage between levels every time they complete a level without dying.



Swinging spike balls

Spikes

In the 2nd layer level ‘Promenade of the Condemned’

Acid pool in a 2nd layer level ‘Toxic Sewers’



Above are two examples from the game. They are two separate levels and have different types of obstacles, in the first one acid pools are a kind of obstacle where if the player stays in them, they take damage over time, whereas in the bottom example, the obstacles distribute instances of damages for each time the player comes into contact with them.

## Essential Features

## Limitations

## Software and Hardware Requirements

## Success Criteria

# Design Stage

## Decomposition

## Structure

## Algorithms / potential solns

## Variables, Data structures, Classes

## Test Data

## Extra

# Implementation Stage

## Stage 1

## Stage 2

## Stage 3

# Testing Stage