ZIGZAG

Game Analysis

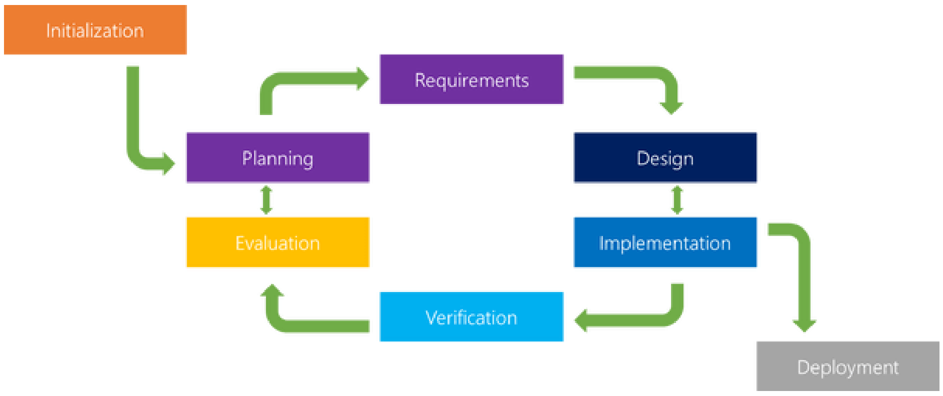
Daniel Cutajar

4/1/2019

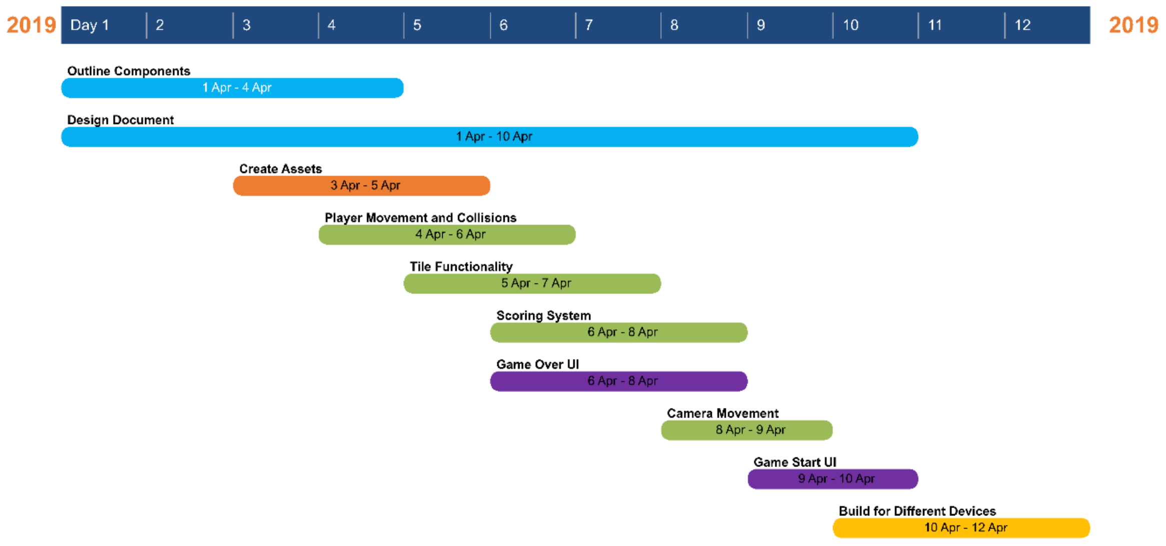
PROJECT TIMELINE & SOFTWARE DEVELOPMENT MODEL

I found an incredibly useful tutorial on YouTube by inScope Studios. The tutorial outlines the development of the basic functionality of an endless runner using ZigZag as its inspiration. This tutorial gave me access to code for implementation along with an understanding of the time taken for each part of the process. InScope Studios' tutorials follow loosely an Iterative Model where he shows the finishes section, explains the requirements to make that section, codes it in and then tests it and implements changes if anything is wrong or missing repeating this process throughout every step in the program.

I plan to follow along with this process as I am following his tutorial. His game does have missing components which I must implement myself after but can also make use of this process.



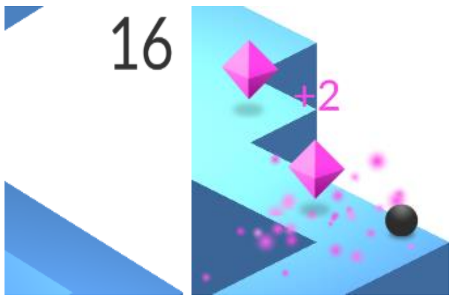
When I need a new feature (I can easily tell what I need by looking at the final game on mobile) first I must plan what I want to add be it camera movement or the Game Start UI and research how to make this work. Afterwards I must note down the requirements, do I need prefabs, UI buttons or a completely new screen. In the design stage I either make the components visuals or decide on the code structure to be used and move on to coding them into the game. I verify by running the game and seeing if it works and evaluating whether I did it well or if things need to be changed. Depending on my answer I either move on to the next phase or go through the cycle again. With this cycle I can also move back to previous phases if I realise I need to update or change a problem found later.



COMPONENTS

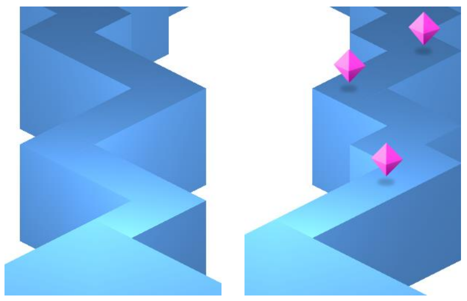
Finding a tutorial on how to make the core components of the ZigZag game on Unity it enabled me to see what different components come into effect how they are gone about in being created.

POINT SYSTEM + MOVEMENT



The game operates on a point scoring system. Whenever the player taps a point is added. This functions by setting the balls direction on start and when tapped direction is switched to another through if statements. In addition, whenever a tap is detected the score is increased by 1. Points are also given from gems which work with collision boxes. If the players collision box collides with that of the gem it shall disappear and register 2 points to the overall score. These scores are stored in the games PlayerPrefs file.

TILE GENERATION

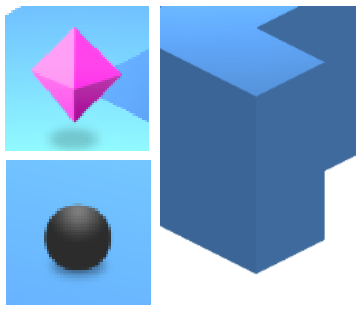


The games paths are made of individual cubes with attach points on the top and left. A random index is used to decide whether a left or top tile is to be spawned and attaches it to the appropriate attach point. This system allows for random path generation but by using an extreme index counter it prevents them from going out of the camera view due to the games camera movement. Pickups are present in all tiles but are deactivated. A random index is also checked before spawning a tile and if the random index is a 0 it activates the pickup to appear on the tile.

FALLING TILES

Tiles also fall behind you. This is not needed for functionality but incredibly helps performance. Tiles are stored in two stacks one for left tiles and one for top tiles. The tile also has colliders that the player passes through and when it leaves the collider, after a fall delay passes the tile will fall downwards and be destroyed after 2 seconds to save memory. These are also replaced in the stack which holds a certain amount and when creating a new tile contacts the stack to see if any are available. This essentially creates reusable tiles to increase performance rather than having new tiles constantly generated as the game goes on.

MODELS



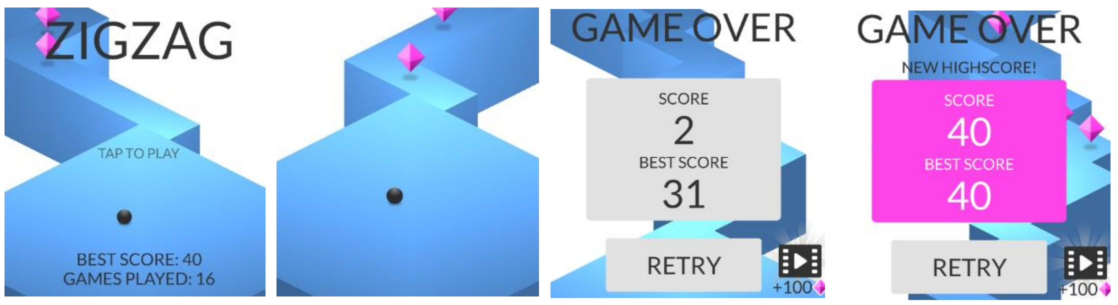
The game uses models stored as prefabs (the player isn't really needed as a prefab). Prefabs are templates which have multiple instances used of them, so the tile and gem are created, stored as a prefab and then called to spawn in the game multiple times but makes use of the prefabs properties throughout. Editing a prefab will edit all instances of that object.

CAMERA

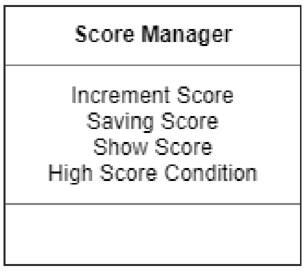
The camera makes use of the players coordinates to move by taking its x and z values combined divided by 2. This calculation essentially causes the camera to move upwards as the player moves but doesn't deviate to the left or to the right. In addition, a spotlight is attached as a child which always remains in the centre of the screen lighting up the path in front of the player. The camera only moves during gameplay and seizes when the player loses.

GAME UI

From the tutorial it is clear to see the game makes use of one scene only. The UI works by constantly being active but through animations moves in and out of the screens view. The game starts with the Game Start UI prompting the player to tap. The moment they do it moves out and the game begins until the player messes up and loses. At this point the Game Over UI comes in from the right showing their score. It has two interfaces depending on whether the player has passed their previous best score or not and switches between these views with a comparative if statement. The retry button essentially resets the game to the Game Start UI closing in again with an endless cycle.

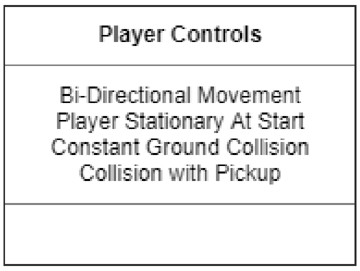


CRC DIAGRAMS



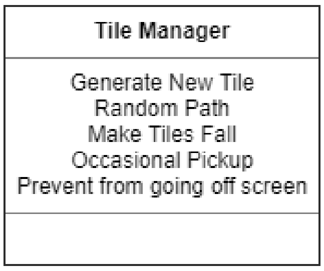
Score Manager

Increment Score
Saving Score
Show Score
High Score Condition



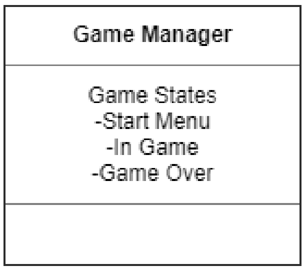
Player Controls

Bi-Directional Movement
Player Stationary At Start
Constant Ground Collision
Collision with Pickup



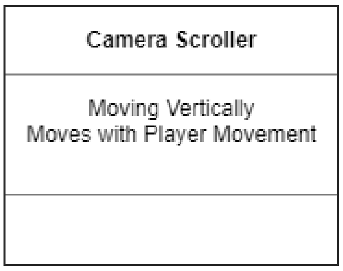
Tile Manager

Generate New Tile
Random Path
Make Tiles Fall
Occasional Pickup
Prevent from going off screen



Game Manager

Game States
-Start Menu
-In Game
-Game Over



Camera Scroller

Moving Vertically
Moves with Player Movement