Introduction

Our primary goal during the planning phase of our video game brief was to capture a modern, recognizable brand that was flexible and approachable to every kind of user. As a result, early discussions shifted away from more traditional forms of arcade gaming such as *Pac-Man* or *Space Invaders*. Instead, group ideas that were shortlisted sought to capture the instantly identifiable nostalgia of games that we, as video game players ourselves, grew up with. Our game finally coalesced into a platforming video game, providing a twist on the original iPhone game *Doodle Jump*. The platforming aspect of *Doodle Jump* provided the perfect combination of simple, easy-to-pick-up game mechanics as well as the ability to implement a diverse range of gameplay twists. In our game you control your ‘doodle’, a customizable player character whose main objective is to steadily climb higher through a randomly generated environment. While doing so, players will encounter several obstacles whose goal is to slow down or defeat the player. These obstacles include different types of platforms such as ghost ledges, which cause the player to fall straight through, or single-use platforms that disappear when stepped on. Obstacles also include active enemies who spawn at varying intervals throughout the map, attempting to end the current user’s game. At the player’s disposal are several powerful abilities or ‘equipment’ that encourage fast-moving platforming, such as spring mechanisms and jetpack boosts. What distinguishes our game from other implementations of platforming games and *Doodle Jump* itself is the introduction of core game-play twists. These include:

* dynamic map manipulation by players that allows for exciting in-game problem solving
* gravity inversion that shifts gameplay away from traditional platforming and towards a race-against-the-clock style
* a competitive two-player version that pits users against each other in an ‘elimination-style’ twist on the Doodle Jump formula

Implementation

Implementing the early platforming logic of the player character and the surrounding objects proved to be relatively straightforward in their design. Writing our game in Processing allowed us to straightforwardly generate player characters, manipulate their velocity, and implement simple collision detection between the user and the surrounding platform. The first challenge that emerged in our game implementation was designing the nature of our randomly generated map, in particular, cycling through successfully navigated parts of the map. One problem that seemed to emerge among other project groups was the difficulty in translating the map and its obstacles to the movement of the user. Our game structure navigated these problems by ensuring that the platform entities (and in turn the equipment and monster entities) were all instantiated within the Doodler class. This allowed us to manipulate these objects in a way that responded to the current configuration of the player character. This also presented a novel solution to a secondary problem that immediately impacted platforming performance and memory allocation. When the map failed to adequately react to the movement of the player the large number of platform, equipment, and monster entities being spawned caused map generation errors, hardware lag, and eventual crashing. By successfully implementing a reactive map generation process, our software was able to identify when parts of the map had been successfully traversed and were therefore being rendered off-screen. In doing so we were able to purge navigated entities that no longer needed to be rendered ensuring a smooth, efficient game execution.

Having implemented a basic platforming method that represented the underlying framework for the rest of our software, the next development challenge was the incorporation of the gameplay twist elements. Developing our two-player game mode presented several challenges, particularly in its integration with the base game model. The two-player method required tweaking how user inputs were registered, allowing for multiple key presses and directional changes among the two users. Creating this alternate game state for two players also required a careful rebalancing of equipment objects. Preliminary feedback recognised that powerful changes to a doodle’s velocity and map position, which caused games to end prematurely, threatened the fairness and enjoyment of the game mode. As a result, alterations were made to interactions with equipment to lessen their impact and the method behind map generation was altered slightly to favour whichever doodle was traversing fastest.

Alongside our two-player game mode, our group endeavoured to develop and refine two other gameplay twists, which also included a reversed gravity mode and the ability for the user to directly manipulate the map layout. Laid out alongside our basic platforming framework, the final challenge required us to combine these disparate packages into one complete product. Important considerations throughout this process were the need to not only create tidy, easy-to-read code for us as developers but an equally accessible user interface for our players. While also remaining keenly aware of how we wanted to structure our game difficulty sections. Our gameplay twists, as well as features such as map enemies, significantly altered the difficulty level of a simple platformer. We, therefore, wanted to create an interface for our game that communicated our gameplay twists to the user, while also allowing for features to be added or removed for certain difficulty levels, all at the discretion of the player. The solution was to implement a layered interface that allowed users to both select single and multiplayer game modes, as well as adjust the difficulty of the map they faced. Subsequently, each incremental difficulty level would enable a new, more challenging gameplay twist.