Planning – 10 marks

Discuss chosen platform – 10marks

Present and discuss the game – 20 marks

Discuss programming approach – 20 marks

Problems and solutions – 10 marks

Reflection – 10 marks

Introduction:

Hi there! I'm Jake Edwards, and I'm excited to share my artefact video for Gameplay Programming. I've created a game called Podium Puzzler, which is a 2D isometric puzzle game. In this game, the player's objective is to move orbs between podiums to unlock new pathways.

Planning:

In the initial stages of my project, I began by outlining the mechanics of my game and how they fit within the adventure genre. My primary focus was on creating a multi-layer A-star pathfinding algorithm that I could use in a puzzle game. Although my game may not be a traditional adventure game, I am confident that the puzzles I have created will provide an exciting and adventurous experience for my players. I designed various puzzle systems, such as interacting with podiums, carrying orbs, and the power lines system, which were essential to making my game possible. To organize my game mechanics, I created a detailed document that broke down the individual components into smaller, manageable problems that needed to be addressed in a specific order. While programming, I discovered that some of the dependent features were not accounted for in my initial plan, requiring me to alter the order of implementation. To avoid this issue in the future, I could have utilized class diagrams. For example, my original plan involved creating the power lines system before implementing the orb movement features, which would have required a rudimentary testing system to debug my power lines. Instead, I restructured my plan to prevent any unnecessary work.

Platform Choices:

After careful consideration, I opted for Unity as the ideal platform for developing my game due to its ability to handle the intricate mechanics I had planned. Having prior experience with Unity, I was confident in its capacity to execute my vision. Moreover, I had previously utilized Unity's Grid and Tilemap system, which I knew would be invaluable in creating my pathfinding. While Unreal Engine presented itself as a viable alternative, I found its layout and functions somewhat complex and challenging to navigate, I lacked the confidence to work with C++ and I dislike Blueprinting. Since I wanted to avoid overwhelming myself with too many unfamiliar elements, I decided against Unreal Engine. While Godot was also a consideration, I had encountered issues with its Yield() functionality in the past, and given that my game would likely require some asynchronous methods, I could not afford to risk any deficiencies in this area. As a result, I ultimately ruled out Godot

.Present Game:

I've created a game that utilizes a complex multi-layer pathfinding algorithm based on A-star. During the development process, I broke down the pathfinding mechanic into various milestones and ensured that the algorithm was fully operational at each stage before proceeding to the next.

To put it simply, the Astar algorithm is an extension of Dijkstra's algorithm that creates a lowest-cost path from the starting position to the end position. It evaluates the traversal cost of adjacent positions and then moves to the lowest cost. It continues this process until it reaches the target position. However, Astar is a heuristic function, which means it is not always the shortest path, but an estimate. It uses estimates to avoid checking all possibilities, unlike Dijkstra's algorithm.

Astar expands on paths that are the least 'expensive' by utilizing an estimated cost from the current position to the goal. Initially, I used the Euclidean Distance Heuristic as the distance heuristic in my algorithm. It calculates the diagonal distance and then the remaining cardinal difference, allowing players to move diagonally as well as cardinal. I accomplished this by multiplying the diagonal distance by an approximation of root 2 to represent the greater distance covered when moving diagonally.

Furthermore, I expanded on this heuristic when I added support for traversing layers, which required using staircases as the only means of moving between layers. To gain a better understanding of the algorithm, I referred to a pseudocode snippet from brilliant.org.

A screenshot of a computer program

Description automatically generated

There are three key mechanics to create the puzzle elements of my game, namely podiums, orbs, and powerlines. Each podium can hold a single orb at a time, and some orbs will spawn alongside them. Four distinct colors of orbs are available, and the player is capable of carrying just one orb at a time. Thus, the challenge rests in strategically placing orbs in the correct location and managing the podiums to ensure access to the necessary orbs.

When an orb of an appropriate color is placed on a podium with a powerline attached, it illuminates and subsequently solidifies or removes specific tiles, thus opening up pathways to additional orbs, podiums, powerlines, or the level's ultimate objective. The concept of orbs can be attributed to the shrine puzzles in Breath of The Wild and Tears of The Kingdom, while the idea of powerlines draws inspiration from Minecraft's Redstone.

Originally, the podiums were meant to open and close areas without the idea of powerlines, but the need for a more easily readable way to show the player which podiums powered specific areas became apparent, particularly as the puzzles became more complex. The concern was that the isometric view is already somewhat confusing initially and adding more complexity on top of that could cause the game to be visually cluttered. However, feedback from friends, family, and peers revealed that few players encountered difficulties in reading the game visually.

One noteworthy piece of feedback was that the player should be able to click directly on the podiums, rather than just the tile beneath them. The implementation of this feature significantly enhanced the game's overall smoothness.

Two additional mechanics that could complement the current mechanics are portals and paintable orbs. Portals could be activated by powerlines and enable the player to teleport through them, carrying the orb with them. Paintable orbs could gain or lose specific colors, and some orbs could be multicolored.

Next steps I would take to polish the game and increase its tactility would be to highlight the tile or podium the mouse is hovering and to display the current path the player is taking. I would also consider allowing the player to cancel their path by right clicking, giving them more time to stop and think if they have made a wrong move.

However, I think the most lucrative expansion on my game, would be a built-in level editor, where players could create their own levels and share them with others, much like in Mario Maker. Players would be able to compete by attempting to complete each other's levels, and a clear rate system could be implemented to determine the levels' difficulty. Additionally, players could have profiles that display medals for each online level they have beaten, as well as their progress in the linear mode. This would help build a community around my game and significantly increase the playtime. It would also add a competitive element and encourage people to share the game with their friends. The online elements could also be expanded into the linear mode, showing completion rates for each standard level. That way you can know how many other players you are better than. And who doesn’t love knowing they’re better than strangers online?

I created 10 levels with increasing difficulty to showcase my mechanics and some appropriate menu screens including a level select and a how-to-play explanation. The player can return to the main menu at any time and for the sake of the gameplay showcase, all the levels are always available.

Programming approach:

Throughout my programming approach, I have used the Graph pattern extensively in the Astar algorithm as it is a path/graph traversal algorithm. I heavily considered using the State pattern throughout my powerlines, orb, and podium system, however, I felt that this would be overcomplicating things when there would only be a few states that could be simply represented by one or two Boolean values and conditions. As I tend to overengineer them into God classes, I chose not to use a Singleton pattern for this project, to attempt to develop more concise reference handling. Another design pattern that I think would have made sense is the Observer pattern. I could have used Actions or Events to signal to the powerlines when to activate and deactivate. This again, however, could be overcomplicating the powerlines and I didn’t explore this pattern in enough detail before planning and developing my game to make an informed decision. I used class inheritance in my powered tiles, which inherits from my wire class and overrides the activate and deactivate functions to interact with the grid. I believe I have also used a Composite pattern in some of the interactions of my astar algorithm such as adjacent nodes. Although I think design patterns are useful and provide important insight into ways to approach a task, I believe that sometimes programmers can rigidly follow them, which can often cause more harm than good when there are logically simpler methods available.

Diving headfirst into my Astar implementation. As I mentioned before I created a few milestones for myself to make this more approachable, I first made my code respect a two-dimensional array of nodes. This was much simpler and just made sure I could implement the Astar model into unity, unfortunately, there isn’t enough time to adequately explain my implementation of the 2d Astar algorithm and I think it is more important to highlight the areas in which I expanded upon it, I will highlight however, that when implementing the basic algorithm, before I set a limit on how many total checks the pathfinding will make before giving up, if I selected somewhere that was impossible to pathfind to, my editor would crash. Teaching me to always be aware of infinite while loops. Anyways, the difficult part was scaling it up to three dimensions. I set this up by creating a CustomTile which inherits from the Unity Tile. This is where I store information about if a tile is ‘layer-traversal’ or ‘walkable’. This is important for enabling the algorithm to recognize where the staircases are. I also had to extend my existing methods to handle a three-dimensional array of nodes and three positions. One of the areas I found most challenging and interesting was finding adjacent nodes. Not only did I need to find adjacent nodes across the current and lower layers when the algorithm was checking a ‘layer-traversal’ tile, but I also needed to check for ‘layer-traversal’ tiles on the layer above regardless of which tile it was selecting. I included a traversal cost of 25 between layers to go with the cost of 10 cardinally and 14 diagonally (an approximation of root 2 multiplied by 10) so that the algorithm will prioritize staying on the same layer when possible. I ran into an issue where the tile being selected was in a position north of the selected position equal to the layer height offset. I solved this by subtracting the z position from the x and y positions when calculating the selected tile. Another oversight was not checking if there was an obstruction on the layer above. This enabled my player to walk through walls. To solve this, I expanded on my validity checks by not only checking if a tile is walkable but also checking if there is a tile above. In my planning, I outlined an extension to my pathfinding which I have implemented. When my player clicks on a podium, I wanted the algorithm to generate a path to an adjacent tile closest to the player. I originally thought I would have to generate a path to all possible tiles and traverse the one with the lowest cost, but I thought this was a brute-force and unnecessarily computationally intense approach. I then thought I could increment each step of the path until a path to one of the adjacent nodes has been found – better but I still thought I was missing something. Then I realized that I was severely overcomplicating things. All I had to do was generate a path to the tile beneath the podium and remove the final step of the path. This perfectly achieves the result I wanted.

Another complex area of my game is the way the paths are traversed. In the past I have used Coroutines to create delayed iteration in my projects, however, I have decided to use asynchronous functions in this project as they are more versatile, the only downside being that as far as I know, OpenGL does not support asynchronous functions, meaning that my game cannot be built for and played in a browser. The use of asynchronous functions over coroutines means that I can have a Task variable that will store the TaskStatus. This means I can check if the function is already running and prevent multiple calls to it. This is important to correctly handle setting a new path while the player is already traversing one. In my first iteration of moving the player toward a target cell, if there was any significant lag, the player would overshoot the destination and continue moving in that direction infinitely. I solved this by updating the direction instead of just calculating it once.

The other elements of my game, although overshadowed by my primary mechanic are complex. The orbs’ ownership is changed when set and picked up, my powerlines use asynchronous functions to light up linearly and the powered tiles enabling and disabling tiles are accounted for in the pathfinding. I have used DontDestroyOnLoad on my loading screen and I load a new scene Asynchronously for the loading screen to fade in and out across scenes. I also included some movement on my loading screen so that if there is a long loading time, perhaps on a weaker machine. The player will know that the game is not frozen.

Module Engagement:

In conclusion, despite facing some challenges at the beginning of the year and struggling to engage with the module due to my EC, I believe that I have significantly developed my skills while creating this artefact. Through this project, I have gained a wealth of knowledge about asynchronous functions, graph and state design patterns, different scopes, and access modifiers such as static and readonly. I have also identified some of my shortcomings and realized that I need to improve my use of property accessor methods and research more design patterns before planning my future projects.

Furthermore, I have conducted extensive research on the Astar algorithm and heuristics, which has expanded my knowledge of advanced programming tasks and provided me with valuable insights into how graph traversal algorithms work. I am proud of the game I have created, despite some missing gameplay loop polish and visual depth.

Thank you.