Dimensions

Part B - Technical Specifications

Rhuta C. Joshi

For my first game project, I have chosen to develop a new game titled “Dimensions”. The following technical specifications provide the necessary details concerning my intended implementation of a classic endless runner game genre with scaling mechanics.

1. GAME DESIGN

a. Player Goals and Objectives

The goal of this game is for the player to manoeuvre and scale a cubical ship in an endless corridor through holes (called windows) on walls blocking the way. This involves navigating through an endless corridor and series of walls facing and obstructing the player. The goal is to collect maximum points and power-ups and to go as far as possible without crashing. The words player and ship are used interchangeably henceforth.

b. Challenges and Conflict

In an attempt to survive without crashing, the player has to move through randomly positioned and scaled windows on each of the incoming walls. Colliding with a wall would result in end of game. The ship’s width and height can be scaled independently to match the window dimensions. The more the shape of the ship matches the window, higher the points.

c. Constraints and boundaries

The player is constrained by several gameplay features. The first of these is the screen area of the game. The player cannot move forward or backward in the corridor. The player is also constrained by the corridor walls, floor and ceiling. The player also has limited time before the next wall appears before which the player has to determine the ship’s position, shape and size. The minimum scale of the ship along its width and height is fixed (the ship cannot go beyond this minimum and vanish).

d. Resources

Players may find some power-ups in some windows. These may help the player with some advantage such as slowing down the walls, freezing everything for some time, shape matching or wall pass throughs.

e. Detailed description of rules, including win/loss conditions

The player ship faces an incoming wall with some windows of different sizes. The ship starts off at a reasonable pace through a linear corridor. As the time in a specific run progresses, the speed of incoming walls and complexity of window positions increases. As the player continues, some power-ups get unlocked and placed randomly. The player loses if the ship touches any of the

corridor walls, floor or ceiling. The number of points collected depends on the extent and number of windows matched in its dimensions.

2. SCENE DESCRIPTIONS

a. Start Screen

The first of these is the startup screen, which allows some limited interactivity for the user to start the game or quit. There is only one startup screen scene to the game, and the animation will loop until an option is chosen.

1. Start Screen Game Object List

i. HighScoreGUI
ii. PlayGUI
iii. ExitGUI
iv. PlayerNameGUI
v. ShipColorSelectorGUI

2. Start Screen Player Input

i. Clicking the StartGameGUI initiates gameplay

b. EndlessRunner Scene

The second scene type is the endless runner corridor scene. There are no levels present in the course of gameplay, the game continues till the player ship crashes. The is where the user manoeuvres the ship position and dimensions as discussed above.

1. EndlessRunner Scene Game Object List

i. ScoreGUI
ii. HiScoreGUI
iii. TimerGUI
iv. PowerUpsGUI
v. ShipObject
vi. WallObject
vii. WindowObject
viii. PowerUpsObject

2. EndlessRunner Scene Player Input

i. W or UP key maps to manual Ship script module that moves the Ship vertically up in the game screen
ii. S or DOWN key maps to manual Ship script module that moves the Ship vertically down in the game screen
iii. D or RIGHT key maps to manual Ship script module that moves the Ship horizontally right in the game screen
iv. A or LEFT key maps to manual Ship script module that moves the Ship horizontally left in the game screen
v. LEFT MOUSE CLICK toggles between width and height selection

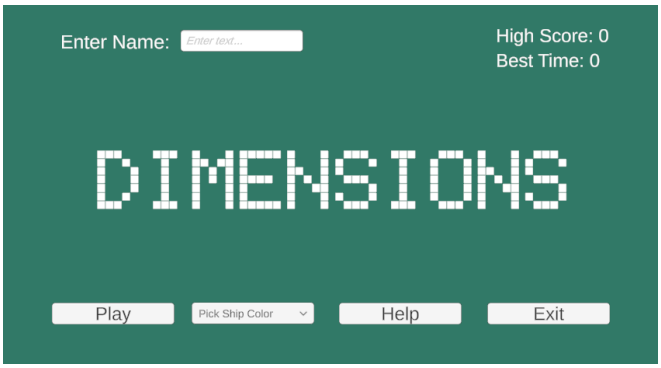
vi. MOUSE SCROLL UP increases the selected dimension (width or height) by a fixed unit
vii. MOUSE SCROLL DOWN decreases the selected dimension (width or height) by a fixed unit

c. GameOver Scene

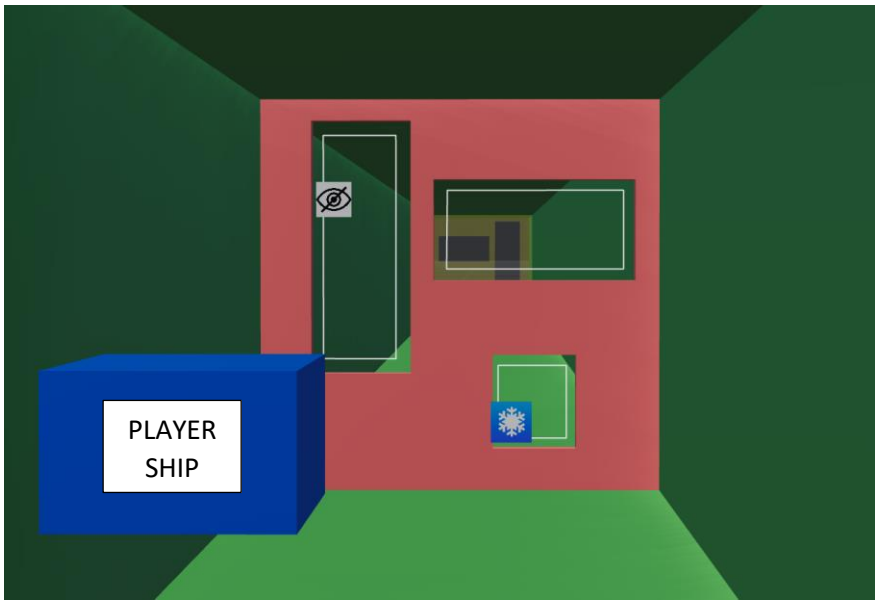
The final scene type is the loss or "game over" screen. Like the start-up screen, there is only one version of this scene, and it is encountered when the user dies and the game is lost.

1. Game Over Screen Object List
    i. GameOverGUI
2. Game Over Screen Player Input
    ii. N/A, short automatic timeout returns to the start screen

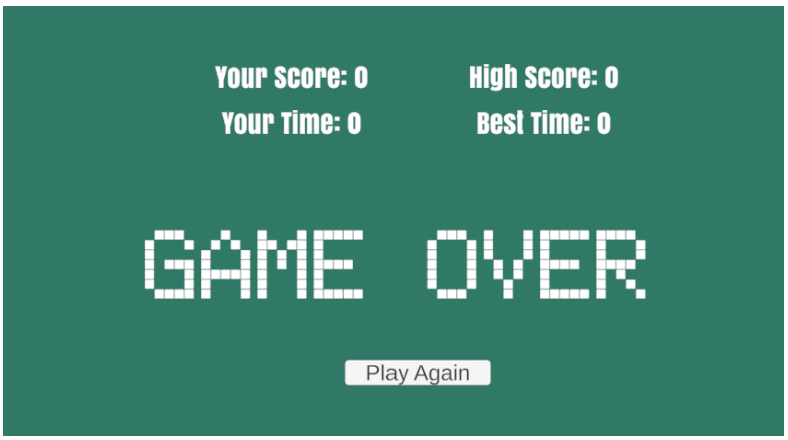
3. SCENE CONNECTIVITY



Start Screen: Welcome, introduction, rules, highscore



*Endless Runner Scene: Player perspective (player is the **blue cube** in this representation)*



Game Over Scene: Spacewalk

4. OBJECT/PREFAB DESCRIPTIONS

a. PlayerPrefab

i. Assets:

1. PlayerRoot – holds player's basic IK chain
2. ShipController – Animator controller and motion controller
3. Materials – Ship's materials
4. ShipAnimations – Idle, flying

ii. Standard Components:

1. Transform: Used to specify the position and orientation of the astronaut character
2. Mesh: The base geometry and renderer of the astronaut character. In my implementation, it will be a mesh cube.
3. Collider: Used to detect collisions and interactions between the ship and the various obstacles and environment units.
4. Rigid Body: Used to set physical rigid body properties

iii. Custom Components:

1. Player (Script):
    a. This script will include the following variables:
        i. alive – this is a boolean variable to determine if the player/ship is alive or not
        ii. speed – this is an integer variable defining the speed with which the ship is moving forward in the corridor
        iii. horizontalInput – this float value will determine how the ship's position changes horizontally on screen
        iv. verticalInput – this float value will determine how the ship's position changes vertically on screen
        v. selectedDim – this is an enum and will select which axis is selected for scaling the ship's dimensions
        vi. scaleInputX – this takes scale input from user to change ship dimensions horizontally
        vii. scaleInputY – this takes scale input from user to change ship dimensions vertically
    b. This script will include the following modules:
        i. Start() – sets up variables listed above
        ii. FixedUpdate() – moves the ship as per user input
        iii. OnCollisionEnter() – detect collisions with other objects in the scene after each move, if it collides with power-ups or is passing through window, points are added to the score, if it collides with anything else, the player dies
        iv. Die() – this function kills the ship and restarts the game

b. CorridorSegmentsPrefab

i. Assets:
   1. FloorPlane
   2. CeilingPlane
   3. CorridorWalls
   4. ObstacleWalls
   5. NextSpawnPoint – an empty game object to determine the next spawn location in loop for endless instancing of segments
   6. Materials – Corridor materials
   7. Audio – audio clips for crash and power-ups

ii. Standard Components:
   1. Transform: Used to specify the position and orientation of the space chamber
   2. Mesh: The base geometry and renderer of the space station. In my implementation, it will be a mesh cube.
   3. Collider: Used to detect collisions and interactions between the ship and the corridor elements.
   4. Rigid Body: Used to set physical rigid body properties

iii. Custom Components
   1. CorridorSegment (Script):
      a. This script will include the following variables:
         i. segmentSpawner – This is reference to a segment spawning manager
         ii. obstacleWallPrefab – This is a game object reference to each incoming obstacle wall
         iii. powerUpPrefab – This is a game object reference to each power-up or points that the player will collect
      b. This script will include the following modules:
         i. Start() – sets up the variables listed above
         ii. OnTriggerExit() – checks when the player exits the collision region, spawns new segment and destroys previous segment
         iii. SpawnWallObstacle() – generates windows and spawns wall in scene
         iv. SpawnPowerUps() – spawns power-ups in windows
   2. SegmentSpawner

c. WallObstaclePrefab

i. Assets:
   1. ObstacleRoot
   2. Windows
   3. Materials

ii. Standard Components:

1. Transform: Used to specify the position and orientation of the space chamber
2. Mesh: The base geometry and renderer of the space station. In my implementation, it will be a mesh cube.
3. Collider: Used to detect collisions and interactions between the ship and the corridor elements.
4. Rigid Body: Used to set physical rigid body properties

iii. Custom Components:
1. WallObstacle (Script) :
a. This script will include following variables:
i. noOfWindows – gives the number of windows in each wall
ii. windows – an array of windows with their respective positions and sizes
b. This script will include the following modules:
i. Start() – sets the above mentioned variables
ii. OnCollisionEnter() – if the player collides with a wall, player dies and if that section of wall is disabled (is a window) nothing happens

d. GlobalObject (GameManager)

i. Assets – None
ii. Standard Components
1. Transform
iii. Custom components
1. GameManager (Script)
a. This script will include the following variables
i. timer – for tracking how long the player has been playing
ii. score – based on scale matching
iii. highscore – overall high score maintained by the game
b. This script will include the following modules:
i. Start() – initializes all variables mentioned above
ii. Update() – called at each time step to update the time dependent global variables listed above

e. HighScoreGUI

i. Assets – None
ii. Standard Components
1. GUIText
2. Transform
iii. Custom Components

1. HighScore (Script) - This script will include a Start() module and an Update() module that communicates with the GlobalObject's global score variable. It will only update the displayed score if it is higher than the GlobalObject's global highscore variable.

f. TimerGUI

i. Assets – None

ii. Standard Components
   1. GUIText
   2. Transform

iii. Custom Components:
   1. Timer (Script): This script will communicate with the Global Object's timer variable to display the time for which the player has survived

g. PowerUpPrefab

5. INTER-OBJECT COMMUNICATION

a. The PlayerObject must communicate with the GlobalScoreObject in order to update the global score variable whenever a move is made

b. The PlayerObject must communicate with all geometry assets in the scene in order to determine the type of geometry during an OnCollisionEnter() event and respond appropriately

c. The TimerGUI object must communicate with the GlobalScoreObject to appropriately display the time to the user via the timer variable

d. The HighScoreGUI object must communicate with the GlobalScoreObject to appropriately display the current score to the user via the score variable