






Roly Poly

Technical Design Document

Team Comet - CSCI356

Name	Number	Email	Contribution
Ashley Collison	5103277	ac641@uowmail.edu.au	100%
Rodney Mampusti	5400879	rm560@uowmail.edu.au	100%
Oliver Rankin-Starcevic	5800559	ors549@uowmail.edu.au	100%
Puneet Talapaneni	5392536	pt614@uowmail.edu.au	100%
Brandon Scolnik	5017737	bs388@uowmail.edu.au	100%

Name	Signatures
Ashley Collison	
Rodney Mampusti	
Oliver Rankin-Starcevic	
Puneet Talapaneni	
Brandon Scolnik	

Target platform and OS	2
Core Game play	2
Game Objects, their properties and behaviours	4
UML Diagram	4
Game Objects	5
PlayerCamera	5
Player	5
Floor	6
Track_Straight, Track_90deg_Curve, Track_Incline, Track_Decline	6
Track	6
StrongAI	6
WeakAI	6
Checkpoint	7
StartPos	7
PowerUp	7
Bomb marble	8
OilSpill	8
Sounds	8
User Interface	9
Main Menu	9
In-Game UI	10
Pause Menu	11
Finish Menu	11
Game Physics	12
Algorithms	12
Physics Materials	13
Artificial Intelligence	14
Algorithms	14
State Machine Diagram	15
Platform specific components	16
Profiling and Optimisation	17
Asset Packages	18

Target platform and OS

Roly Poly is a mobile app for smartphones running on Android. Minimum Android version: Android 4.1 'Jelly Bean'.

Core Game play

Roly Poly is a physics based marble racing game where players race marbles around a track. Players have the option to customise their marble size and mass, as well as colour for aestheticity. Smaller marbles have a faster acceleration but have a slower maximum speed and are more susceptible to being knocked off course by other marbles. Lighter marbles have a faster acceleration but have a slower maximum speed and are more susceptible to being knocked off course by other marbles. Heavier marbles have a slower acceleration but can reach a higher maximum speed and are less susceptible to being knocked off course by other marbles. Larger marbles have a bigger hitbox, allowing them to knock other players off course and get pick ups easier. However, they will have a harder time avoiding obstacles around the track. The Players will have to race other players around a map full of various obstacles just as walls, cannons and moving platforms. Races can be single player against AI. Initially it was planned to have online multiplayer against human opponents, however this was cut due to time constraints. Scattered across the track will also be powerups which players can use to gain advantages, or give other racers disadvantages. After each race the player will get to see what position they finished the race in.

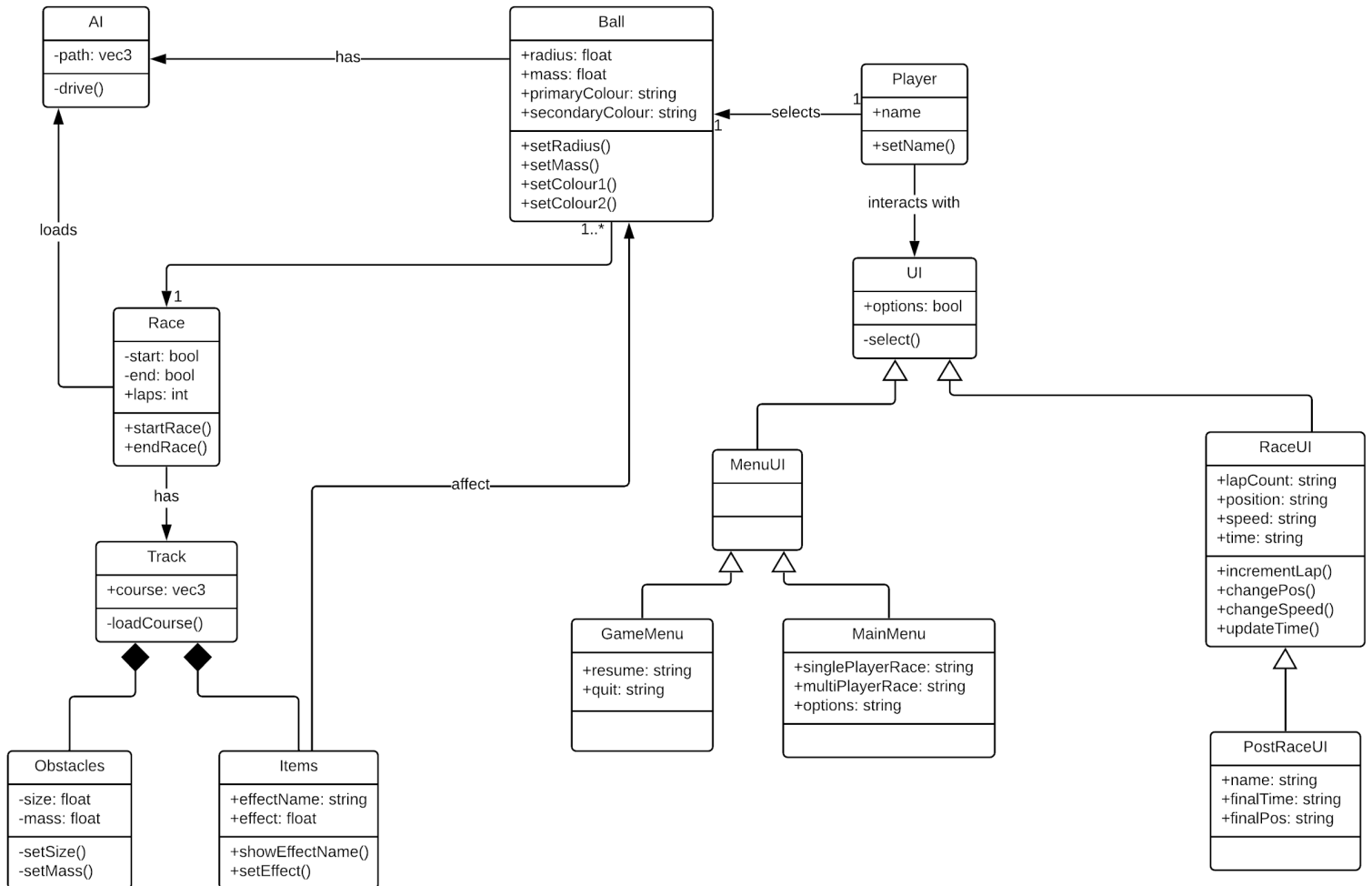
CSCI 356 Game engine essentials - Group assignment - Team Comet

The Powerups:

Name	Effect
Oil slick	Leaves a pool of oil behind the racer. Slows any racer who collides with it.
Bomb	Leaves a bomb behind the racer. Pushes players away from it after detonation
Speed boost	Double acceleration of racer
Strength boost	Increases the mass, acceleration and size of the racer.

Game Objects, their properties and behaviours

UML Diagram



Game Objects

PlayerCamera

Follows the player and acts as the view for the user.

Components:

- Tag: MainCamera
- Camera: 90 FOV
- Skybox: sky02 from ClassicSkyboxes
- CameraController: Custom script which positions the camera behind the player, based on the normalized velocity as the direction of the players movement. Also uses lookRotation to look at the player. Makes use of the Slerp functions to allow for smooth movement and adjustment of camera position and direction.

Player

The object which the user controls in the game.

Components:

- Tag: Player
- Transform: Scale is modified by the user from the main menu.
- Rigidbody: Mass is modified by the user from the main menu. Uses gravity.
- SphereCollider: Uses marbleMaterial, a custom material with 0.5 bounciness, 0.3 friction, average combines.
- Controller: Custom script used to take input from the user and apply movement to the player. MaxSpeed and Acceleration are set based on the players configuration of their mass, in order to balance the game. A check is made that the players current speed is not greater than maxSpeed, and if so, it gets a move vector based on the user input (either by gyroscopic controls or by the joystick) and transforms the direction of the movement based on the current way the camera is facing. Then a force in the direction of the vector produced is applied to the player.
- LoadGamePrefs: This file retrieves the playerPrefs values that were set by the player in the main menu, and applied them to the game. marble size, mass colour and specular colour is set, music or sound is updated to the correct value in the pause menu when accessed, and the bump sound effect is also added to the players marble.
- TrackProgress: This tracks the players overall progress on the track, and updates the progress bar ui element accordingly. The progress is measured as a cumulative value based on the distance from checkpoint to checkpoint.
- TrackPosition: Track position is similar to trackProgress, but it measures the progress of all other AI and then returns the players ranking in relation to the other AI's progress. Also features an onTriggerEnter which places the player back to their last checkpoint in the event that they fall off the track.
- Shader: Uses marbleColour, a standard specular shader which features a stripy pattern albedo and the colours chosen by the user.

Floor

The floor of the world space, if a marble falls off of the track, it will collide with this floor and respawn at their last checkpoint.

Components:

- Tag: Respawn
- Collider:IsTrigger to trigger a respawn.

Track_Straight, Track_90deg_Curve, Track_Incline, Track_Decline

- Bumpers: Prevents marbles from rolling off the edge of the floors in areas where that may be difficult to avoid. Uses the railing albedo.
- Floors: The ground on which the marbles shall roll. Uses the Track albedo.
- SeamFix: A collider with an attached script used to prevent players from bumping over the seams between tracks. It does this by freezing y position while passing over these seams.

Track

The physical structure upon which the marbles move. It has a NavMesh layered on top so that the NavMeshAgent AI can move around it.

- Children: wp1-wp7
Designated locations on the track where a checkpoint will be placed. A checkpoint moves to the next location when its corresponding racer collides with it.

Components:

- Tag: waypoint

StrongAI

Prefab defining the stronger type of AI. It is coloured with a red material to designate this. It is a NavMesh agent with the checkpoint as its target. When it collides with the checkpoint, the checkpoint moves to the next location on the track and the agent recalculates the route to it within the NavMesh.

Components:

- Tag: racer0 - racer7 (initialised on game start)
- NavMeshAgent: To distinguish from the weaker AI, the NavMeshAgent's acceleration is given a higher possible range.
- Rigidbody: Uses gravity.
- AIRacers: A script which sets the target for the NavMesh agent based on its tag.
- AIPowerUpPickup: A script which sets the radius and angle in which the racer will detect power ups and seek them out. To distinguish from the weaker AI the radius and angle were increased.

WeakAI

Prefab defining the stronger type of AI. It is coloured with a red material to designate this. It is a NavMesh agent with the checkpoint as its target. When it collides with the checkpoint, the

checkpoint moves to the next location on the track and the agent recalculates the route to it within the NavMesh.

Components:

- Tag: racer0 - racer7 (initialised on game start)
- NavMeshAgent: To distinguish from the weaker AI, the NavMeshAgent's acceleration is given a lower possible range.
- Rigidbody: Uses gravity.
- AIRacers: A script which sets the target for the NavMesh agent based on its tag.
- AIPowerUpPickup: A script which sets the radius and angle in which the racer will detect power ups and seek them out. To distinguish from the weaker AI the radius and angle were decreased.

Checkpoint

Prefab defining the target that the NavMesh agent and the player moves towards in order to progress the race. When collision with either an agent or the player occurs the checkpoint disables its collider and moves to the next designated location on the track, then reenables its collider. The last checkpoint reached is saved as the respawn point.

Components:

- Tag: playerMarker, Marker0 - Marker7 (playerMarker defines the target for the player, Marker defines the targets for the AI, initialised at game start)
- Collider: IsTrigger is used to detect collision and to move the checkpoint to the next location.

StartPos

Prefab defining the starting position for the AI. Instantiated at game start depending on how many AI are selected to race against. Instantiated on the track before the AI with their positions being used as the Instantiation positions for the AI.

PowerUp

This object can be picked up by any racer and will give a buff to them, or a debuff to those around them.

Components:

- Tag: Power
- Layer: Power
- PowerUps: A script which handles all things to do with powerups. It plays the corresponding sound of each powerup, rotates the powerups upon the y axis, and detects collisions. Upon collision it will determine whether an AI or the user has collided with it. For both AI and user it will determine whether that racer is in last place, and if so that racer is guaranteed to receive powerup4. If not, the racer will randomly receive powerup 1, 2 or 3. For the player, the powerup is displayed as a button icon, which the user can press to activate the powerup. For the AI, there will be a random waiting period of 0-5 seconds before it automatically activates the powerup. For each powerup function, the results are the same, however the methods are different (e.g. modifying Controller for player, and NavMeshAgent for AI).

Bomb marble

Properties of the bomb marble are handled in the PowerUps script. A bouncy black marble is instantiated which then, after a short period, is destroyed and an explosion is instantiated in its place.

Components:

- Rigidbody: Uses gravity.
- SphereCollider: Uses BombMaterial, which makes the bomb bounce.

OilSpill

A Flattened sphere with an oily look which slows down racers who collide with it and then is destroyed upon the racers exit.

Components:

- Rigidbody: Uses gravity.
- BoxCollider: This is to allow it to look flat on ramps (while using gravity).
- SphereCollider: IsTrigger, triggers the OilSlow script.
- OilSlow: Custom script which checks if a player or AI has collided with it, and adds a force in the opposite direction to their velocity, thus slowing them down. Destroys parent upon trigger exit.
- Shader: Shader with oil slick albedo and normals which was sourced from mega shader set asset set.

Sounds

Children: Each child is an object containing one AudioSource component. This allows for audio to be activated and deactivated easily.

User Interface

Main Menu



The first screen that the player sees, this menu gives the player some customizability options. This GUI (and all following GUIs in this project) use the **Canvas Scaler** component, which maintains the size of the UI regardless of screen resolution. Navigation between the menus is done by using **Button** components with **OnClick** events which **SetActive(false)** the currently open menu, and **SetActive(true)** the menu which is to be opened from the button being pressed. **PopSound** component **AudioSource** is also played on button click. The **MenuMarble** object symbolises the player, and serves the purpose of aestheticity and customisation feedback. It also has a **MenuMarbleMat** physics material which allows it to be bouncier.

All preferences set by the user are saved using the **PlayerPrefs** class, which allows for key value local storage. This is done in the **LoadMenuPrefs** script. This script also updates the **MenuMarble** to match the user's preferences. In the top right there are **Toggle** buttons which can toggle music, sound and gyroscopic controls. Script **OnLoad3** initializes these toggles to match saved user preferences on startup. Using **TextMeshPro** from the asset store allowed for better looking fonts to be used, as well as gradient colour of text. Two custom gradients that have been used are **Fire Gradient** and **Title Colour**.

The **MY marble** menu is split into two sections, stats and style. Stats allows the user to change marble mass and size, and style allows for colour and specular colour to be modified. Both menus use **Slider** components between appropriate max and min limits. **OnLoad1** and **OnLoad2** initialize player stats and style sliders respectively to match saved user preferences

on startup. The user can also start the game from the main menu, but must first set number of AI and AI difficulty, which also uses sliders initialized by **OnLoad4**. When the user clicks GO, the **Go** function in the script **MainMenu** is called, which loads the scene **Final Track**.

In-Game UI



LoadGamePrefs script is used to load all user settings that were set in the main menu, into the game (marble size, mass, colour, and music, sound and gyro toggles).

In the top right is the pause button, This button sets active the pause menu.

At the very top is the progress bar. This is a slider which the user cannot interact with. It fills depending on the total progress of the player, with a max value of the distance of the entire race, using the **TrackProgress** script.

The lap in the top left is also tracked using the **TrackProgress** script, and is stored as an int starting at one and incremented using the **IncrementLap** function, which is called upon in the **TrackMarker** script every time the player reaches the final waypoint and invokes the first waypoint to loop back as the next waypoint. When lap is 4, the finish menu is set to active.

Also in the top left is the players position, which is tracked in **TrackPosition**, which calculates the AI's progress and ranks the player's progress relative to them. Then an if else if determines whether to display 1st, 2nd, 3rd or nth.

In the center of the screen is the countdown text. This text is set to "ready", "3", "2", "1", "go" with a beep to count the player in before controls are activated and AI is spawned in. This is done in the **CountDown** script.

The bottom left is the mobile joystick. Note that this is not active if gyroscopic controls are toggled on (**OnLoad3**). This controls the direction and magnitude of force that is applied to the player, and is responsible for the players movement. The **Controller** script checks the users input, and based on camera direction, will apply force appropriately.

At the bottom right is the activate powerup button. It is just an image and does not have button functionality until a powerup is picked up. Upon the



player picking up a powerup, the **PowerUps** script will set active one of four buttons, which has a unique icon and will grant an ability on click.

Pause Menu



Uses the **OnLoad3** script to initialize values of the toggle buttons, and also sets timescale to 0, pausing the game. The user may use the toggle buttons, return to their game or return to the main menu. Returning to the main menu uses the **Back to menu** function in the **MainMenu** script.

Finish Menu



Uses the **OnLoad3** script to set timescale to 0. The user can only go back to the main menu from this screen (in the same way as pause).

Game Physics

Algorithms

Name	Description	Implementation
Movement - Input manager	Player moves the on-screen joystick to move their marble	Script uses <code>CrossPlatformInputManager.GetAxis()</code> to grab the Horizontal and vertical input from the on screen joystick. The script then uses these components in the vector 3 called "movement". This movement vector is normalized and multiplied by a direction value to get the amount of force added to the marble depending on the direction which the player camera is facing. Also the force is multiplied by the acceleration value, which set based on the player marbles mass. If the player's velocity magnitude is greater than the <code>maxSpeed</code> value, no force is applied to the marble.
Movement - Tilt controls	Player tilts the device to move their marble	Script uses the input acceleration from the device in the vector 3 called "movement". This movement vector is normalized and multiplied by a direction value to get the amount of force added to the marble. The base input acceleration is offset by 90 degrees to allow the user to comfortably use the device on a flat surface
LoadGamePrefs	Players customisation influences the size and mass of the marble.	A larger marble size means that there is more surface area on the marble, and therefore one rotation moves the marble further than a single rotation from a smaller marble. A bigger marble also means that there will be less area for the marble to navigate, making handling more difficult. In order to balance the game, when the player has a higher mass, they also receive a higher acceleration in the Controller script. Higher mass still causes the marble to have less speed than a lighter marble, due to it requiring more force to reach the same velocity as a lighter marble. This means that handling is harder for a larger marble as well, since applying a force in a new direction will go against the marble's momentum.
Explode	A bomb powerup that	Applies an impulse force of magnitude 50 in a radius

	applies force to marbles in an outward direction from the point of detonation.	of 5 units from the point of explosion.
OilSlow	An oil slick debuff which applies a force in the opposite direction of current velocity.	Applies a force relational to the racers current velocity, which is in the opposite direction of the velocity multiplied by 50. This causes those who collide with it to slow down.
Collision detection	AI collision with checkpoint markers.	Upon collision with a checkpoint marker the marker's collider is disabled, allowing the AI to move unobstructed. The marker's collider is then reenabled when it moves to the next section of the track.

Physics Materials

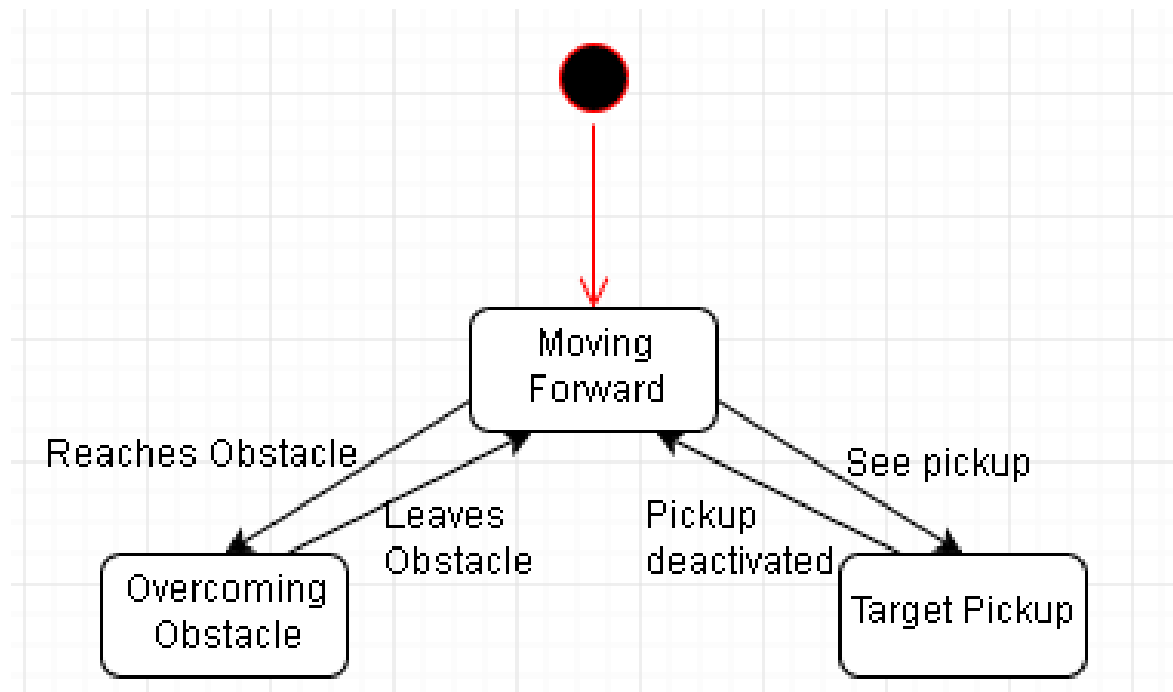
Name	Properties
marbleMaterial	Used for the player marble while in game. Has 0.3 friction static and dynamic, 0.5 bounciness, and average bounce combine. This will give the marble some bounce, but will not be much bounce, similar to the material of a billiard marble.
BombMaterial	Used for the bomb marble object, this material has 0.85 bounciness and maximum bounce combine, meaning it will be quite bouncy. This object will roll a short distance before exploding.
marbleFriction	
MenuMarbleMaterial	Material used for the marble in the main menu. This marble is bouncier for aesthetic reasons. Maximum bounce combine, 0.5 bounciness, 0.3 friction (static and dynamic).

Artificial Intelligence

Algorithms

Name	Description	Implementation
AIPowerUpPickup	AI marble uses pickups to sabotage another marble or boost itself ahead.	All power ups picked up by the AI are activated within 0-5 seconds.
AIRacers	AI marble moves in a forward direction along the path to move towards the finish line.	Invisible waypoints are placed along the top of the track, marking points of direction for the AI. Once the AI has reached a waypoint it moves toward the next. In the event of being reset to the nearest checkpoint, the first waypoint beyond the checkpoint must become the AI's directive.
TrackMarker	An invisible object that indicates where the AI should go next.	TrackMarker is set as the destination for a NavMeshAgent. When the Agent collides with the track marker, the marker disables it's collider, moves to the next checkpoint on the track and reenables it's collider.
RaceStart	Instantiates AI at the start of the race based on player preferences.	Reads player settings for number of AI and difficulty. Depending on the settings, instantiates 1-8 players and easy and/or hard AI at the starting line.

State Machine Diagram



Platform specific components

Compatibility

- Compatible with Android 4.1 'Jelly Bean'

Controls

- Tilt controls and joystick controls allows the player to control the game in a manner which is most viable for them
- Toggle option that allows the player to switch between the two controls

Project Quality Settings

- "Fastest" or "Fast" quality levels
 - Game will be played on mobile, higher levels will make little difference in terms of visual quality on a small screen
 - Focus on performance
- Shadows properties
 - Shadows: "Hard Shadows Only"
 - To simulate shadows when the marble is rolling on flat track, going up or down inclines and when it's airborne
 -
 - Shadow Resolution: "Medium"
 - The lowest setting that gives the marble's shadow a smooth definition
- Vsync - Off
 - Screen tearing during gameplay is minimal
 - Reduces input lag

Testing - Build to android testing device

- Testing builds were created and downloaded to an android device running colorOS V3.0.0i for testing as this device is the one we were using for demonstrations.

Profiling and Optimisation

Accelerometer Optimisation

Since our game utilizes an accelerometer we looked into optimizing the accelerometer sampling, since by default unity samples the accelerometer at 60 hertz. However we discovered that

Shadow Optimisation

Our game only uses hard shadows with medium resolution as this improves performance of the game on mobile devices.

Baked Lighting

In order to maximise frames per second while using the editor, we temporarily disabled global illumination baking whilst working on the project. Once it was complete, we reactivated it.

LOD

As the models were primarily primitives, and the textures basic, we did not see it as a crucial task to handle LOD.

Asset Packages

- Standard Assets
- Mega Shader Set
- Ball Pack
- DigitalKonstrukt
- Sci-fi Styled Modular Pack
- TextMesh Pro
- Classic Skybox