

Road-bound Legends

Unity 6 Based Motocross Racing Game

Shaymaa Soufan 202303528

Omaima Marmar 202303330

Ahmad Fakih 202300230

Hassan Fakih 202302716

Mohammed Mtayrek 202306092

Ahmad Hassan 202306066

**Presented to: Dr. Imane Haidar COMP442: SoftwareEngineering**

1

Contents

[Project Overview: 4](#_Toc196782434)

[Objectives: 4](#_Toc196782435)

[Background: 5](#_Toc196782436)

[Literature Review: 5](#_Toc196782437)

[Conclusion: 9](#_Toc196782438)

[Project Planning: 9](#_Toc196782439)

[I. Implementation Environment of the Current System: 9](#_Toc196782440)

[II. Minimum System Requirements: 10](#_Toc196782441)

[III. Recommended System Requirements: 10](#_Toc196782442)

[IV. Off-the-shelf Software: 10](#_Toc196782443)

[V. Anticipated Workplace Environment: 11](#_Toc196782444)

[VI. Schedule Constraints: 11](#_Toc196782445)

[VII. Budget Constraints: 11](#_Toc196782446)

[VIII. Project Issues: 12](#_Toc196782447)

[IX. Migration to the New Product: 12](#_Toc196782448)

[X. Risks: 12](#_Toc196782449)

[XI. Team Members’ Tasks: 13](#_Toc196782450)

[XII. Ethical Issues: 13](#_Toc196782451)

[XIII. Software Model Process: 14](#_Toc196782452)

[XIV. Feasibility Study: 14](#_Toc196782453)

[XV. Standards: 14](#_Toc196782454)

[XVI. Milestone: 15](#_Toc196782455)

[Citations: 16](#_Toc196782456)

[Requirements: 17](#_Toc196782457)

[Use Case Diagram – “Road Bound Legend” - Game Application 17](#_Toc196782458)

[Functional Requirements: 19](#_Toc196782459)

[Data Requirements: 20](#_Toc196782460)

[Non-Functional Requirements: 20](#_Toc196782461)

[Design: 22](#_Toc196782462)

[ System Architecture: 22](#_Toc196782463)

[ User Interface: 22](#_Toc196782464)

[ Class Table: 23](#_Toc196782465)

[ Variable Table: 23](#_Toc196782466)

[ UI WireFrames: 24](#_Toc196782467)

[ Activity Diagram: 26](#_Toc196782468)

[ Tools And Technologies: 26](#_Toc196782469)

[ Coding Standards: 26](#_Toc196782470)

[Implementation: 27](#_Toc196782471)

[Testing 31](#_Toc196782472)

# Project Overview:

**“Road-bound Legends”** is a 3D realistic physics racing & adventure game designed and programmed using **Unity 6 engine**. Players will experience adventurous high-speed motocross racing, explore our interactive world including vast environments. The game offers dynamic gameplay and easy to learn mechanics leading to an unforgettable riding experience. The project serves as both an entertaining racing experience and a learning exercise for software engineering, with a focus on gameplay design, environment construction, and optimization.

# Objectives:

* Develop a 3D motocross game in Unity 6 with easy to play experience
* Designing wide & vast environments featuring different time-of-day and weather settings.
* Integrate realistic physics simulation and smooth turning mechanics.
* Implementing machine learning agents that learn, adapt, and maneuver smoothly throughout the designed environments

# Background:

Dirt-Bike racing games have come a long way, from basic 2D designs to detailed 3D worlds. Today, they attract both casual and serious gamers. With the popularity of mobile and easy-to-play games, there’s a demand for racing games that are simple yet fun. Features like customizable skin, dynamic environments, day-night cycles, and easy controls make games more engaging.

But motocross isn't just about crazy graphics it’s about the experience. Realistic physics brings weight to every jump, bump, and landing. ML agents (AI powered opponents) can create smarter opponents that adjust to player gameplay, guaranteeing that each new race brings a whole new experience. This review looks at how these features can be combined to create a fun and accessible 3D motorcycle racing game for everyone

# Literature Review:

The purpose of this literature review is to analyze similar products (web, apps, and other games) and research papers from both Lebanon and other countries, focusing on motocross racing games and relevant technologies. Each product or research is evaluated based on its description, advantages, and problems. A table at the end summarizes these aspects along with the references.

Similar Products and Research Papers:

1. Road Rash (App):

A popular racing game with a focus on combat mechanics. Players ride motorcycles while fighting opponents to win races. It is known for its fast- paced gameplay and dynamic combat system. However, the game lacks realistic physics and often feels arcade-like rather than immersive.

1. Motocross Madness (PC Game):

An older motocross racing game that offers impressive terrain simulation and stunt mechanics. The game is praised for its physics-based gameplay and open environments but lacks modern graphical fidelity and AI advancements seen in contemporary games.

1. Rider (Mobile App):

A simple 2D motocross game where players perform stunts while riding on a physics-based track. It is highly addictive but lacks realism and depth compared to 3D motocross games.

1. Microtransactions in Gaming (ResearchGate, Serbia, UK):

Microtransactions have become a prevalent monetization strategy in video games, allowing players to purchase virtual goods or services. Research has

identified both positive and negative implications of this model. A systematic review found that microtransactions, particularly loot boxes, are associated with gaming and gambling disorders, raising concerns about ethical implementation. From an industry perspective, microtransactions have transformed business models, leading to increased revenue streams but also sparking debates about their impact on game design.

1. AI and Machine Learning in Racing Games (ResearchGate, NVidia):

Artificial Intelligence (AI) plays a significant role in modern racing games, particularly in providing realistic opponent behavior. Traditional AI methods, such as waypoint-based pathfinding and behavior trees, have been used to create dynamic opponents. Recent advancements have introduced reinforcement learning techniques, where AI agents learn from experience to navigate complex racetracks, optimizing lap times and adapting to player behavior.

1. Internet Gaming Disorder in Lebanon (Journal of Behavioral Addictions):

This study investigates the prevalence of IGD among Lebanese adolescents, highlighting associations with age, reduced sleep, and lower academic performance. The research focuses on high school students and emphasizes the impact of online gaming on youth behavior.

Summary Table:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Ref No.** | **Authors/Source** | **Description** | **Advantages** | **Problems** |
| 1 | Road Rash (App) | Combat-focused motocross racing game | Fast-paced gameplay, dynamic combat | Lack of realism, arcade-like physics |
| 2 | Motocross Madness (PC Game) | 3D motocross racing with terrain simulation | Realistic physics, stunt mechanics | Outdated graphics, limited AI capability |
| 3 | Rider (Mobile App) | 2D stunt-based motocross racing game | Simple, addictive gameplay | Lack of depth, unrealistic physics |
| 4 | Nenad Tomić, University of Kragujevac | Effects of microtransactions on the video game industry | Financial benefits for developers | Potential for gambling behavior |
| 5 | Peter Edwards, NVidia On Demand | Training AI agents to race using reinforcement learning | Dynamic and adaptive AI | Computational intensity |
| 6 | Erin Gibson et al., Nottingham Trent Uni. | Video game player experiences with microtransactions | Insights into player dissatisfaction | Monetization reduces satisfaction |
| 7 | Nazir S. Hawi et al., Journal of Behavioral Addictions | Internet gaming disorder among Lebanese youth | Insight into IGD prevalence and related factors | Focus on IGD rather than general trends |

# Conclusion:

The development of *Road-bound Legends* aligns with current research on game development best practices. Leveraging Unity 6’s advanced features, integrating physics-based simulations, AI-driven opponents, and procedurally generated environments ensures a dynamic and engaging experience. Additionally, ethical considerations surrounding microtransactions highlight the importance of designing monetization strategies that enhance rather than hinder player satisfaction.

This literature review provides a solid foundation for the game’s development, ensuring that industry standards and research-backed methodologies are applied effectively.

# Project Planning:

Constraints:

## Implementation Environment of the Current System:

The motorcycle game will be developed using **Unity 6**, leveraging the **Unity Asset Store** for pre-made assets to assist with physics and environment building. This approach will significantly reduce development time while ensuring quality and realism in gameplay and keeping game in line with alight system requirements.

## Minimum System Requirements:

* + **Processor:** Intel Core i5 or AMD equivalent
  + **Graphics:** NVIDIA GTX 970 or AMD R9 390
  + **RAM:** 8 GB
  + **Storage:** 10 GB available space
  + **Operating System:** Windows 10 or macOS Mojave

## Recommended System Requirements:

* + **Processor:** Intel Core i7 or AMD Ryzen 7
  + **Graphics:** NVIDIA GTX 1080 or AMD RX 5700
  + **RAM:** 16 GB
  + **Storage:** 20 GB available space
  + **Operating System:** Windows 11 or macOS Monterey

## Off-the-shelf Software:

The project will utilize the following software:

* + **Unity 6**: Game development platform
  + **Unity Asset Store**: Pre-made assets for physics and environment design
  + **Blender**: for 3d designs.
  + **Git:** Used for version control and source code

## Anticipated Workplace Environment:

The project will be developed primarily in a **university lab**, with additional work completed on **personal laptops**. Access to the lab is granted by the university, allowing for collaboration and testing in a controlled environment. Unity easy to run requirements benefit us as the project could be worked on from any device with relatively medium to high end specs.

## Schedule Constraints:

The project timeline is set to **12 weeks**. Additionally, scheduling must account for **lab availability** and coordination among group members.

## Budget Constraints:

The project operates on a **low budget**, with expenses mainly directed toward **paid tools and assets from the Unity Asset Store,** for the future a higher budget is needed mainly for server allocation, Anti-cheat software and extra development cost .

## Project Issues:

Currently, the primary issue is **finding a game publisher to release the game under**. Which is important in the long run since it gives us access to proprietary software not released to the public and huge resources like bigger budget and team, this matter remains unresolved and will be addressed as development progresses.

## Migration to the New Product:

Migration will involve **updating and integrating new assets or features** to align with modern gaming standards and player expectations, in addition to releasing the game on more platforms other than windows (mobile, game console…).

## Risks:

The key risks include:

* + **Budget Overruns**: Due to unforeseen software costs
  + **Time Constraints**: Managing the 12-week deadline efficiently
  + **Technical Challenges**: Implementing realistic physics and optimized performance
  + **Release Challenges**: Securing a game studio for publication

## Team Members’ Tasks:

* + **Manager**: Project oversight, timeline management, and resource allocation.
  + **Designer**: Environment creation, asset integration, and visual consistency.
  + **Developer**: Coding mechanics, physics implementation, and debugging.

## Ethical Issues:

Potential ethical issues Primarily include **Microtransactions**, and age ratings also less prominent issues include fair use of assets, data privacy for users and maintaining accessibility standards.

Microtransaction can be seen as exploitative as some say it preys on little children by using colorful skins and loved collaboration, in some countries a form of microtransaction (loot boxes) is considered gambling and treated as one legally.

The age limits most motor games as 7+ but some argue it is too low, and this type of game inspires kids to replicate movies seen on games.

## Software Model Process:

The project will follow an **Agile development methodology**, focusing on iterative improvements and regular feedback.

Reason for choosing the agile methodology:

1. Project that **requires a prototype** before working on the final version
2. Project **require multiple tasks** that require strong collaboration
3. Project has **no specific outcome** or vague requirements

## Feasibility Study:

The feasibility of the project is supported by:

* + **Technical Feasibility**: Unity’s robust environment for game development
  + **Operational Feasibility**: Accessible lab environment and personal laptops
  + **Economic Feasibility**: Low-budget approach with minimal software costs

## Standards:

* + **Coding Standards**: Consistent naming conventions and modular code structure
  + **Art and Design Standards**: Realistic environments and accurate physics
  + **Testing Standards**: Regular bug tracking and performance testing

## Milestone:

* + **Week 1-2**: Planning and initial asset collection
  + **Week 3-6**: Environment and physics implementation
  + **Week 7-9**: Game mechanics and feature integration
  + **Week 10-11**: Testing and bug fixing, adding game menu
  + **Week 12**: Final Review and presentation

# Citations:

* ***Benoit Chaperot & C. Fyfe , University of Burgos, Motocross and artificial neural networks , Research Gate****. (July 2008). Retrieved from https://*[*www.researchgate.net/publication/228962866\_Motocross\_and\_artificial\_neural\_netwo*](http://www.researchgate.net/publication/228962866_Motocross_and_artificial_neural_netwo) *rks?*
* ***Erin Gibson, Mark D Griffiths, Filipa Calado, Andrew Harris, Nottingham Trent University***

***,Videogame player experiences with micro-transactions: An interpretative phenomenological analysis, ResearchGate.*** *(2023, Aug). Retrieved from https://*[*www.researchgate.net/publication/369568354\_Videogame\_player\_experiences\_with\_m*](http://www.researchgate.net/publication/369568354_Videogame_player_experiences_with_m) *icro-transactions\_An\_interpretative\_phenomenological\_analysis*

* ***Martin Best, Unity Blog , Unity 6 is here: See what's new. (2024, Oct 17)****. Retrieved from https://unity.com/blog/unity-6-features-announcement*
* ***Nenad Tomić, University of Kragujevac , Effects of micro transactions on video games industry, ResearchGate. (Jan 2017). Effects of micro transactions on video games industry.*** *Retrieved from https://*[*www.researchgate.net/publication/322313479\_Effects\_of\_micro\_transactions\_on\_vide*](http://www.researchgate.net/publication/322313479_Effects_of_micro_transactions_on_vide) *o\_games\_industry*
* ***Peter Edwards, Machine Learning in Games: Training AI Agents to Race Using Reinforcement Learning, NVidia On Demand. (2021, April).*** *Retrieved from https://*[*www.nvidia.com/en-us/on-*](http://www.nvidia.com/en-us/on-) *demand/session/gtcspring21-e32776/*
* ***Unity , Campaign , Resources , Unity 6 for games. (2024).*** *Retrieved from https://unity.com/campaign/unity-6-resources*
* ***Unity , Campaign , Unity 6 for games. (2024)****. Retrieved from https://unity.com/campaign/unity-6-for-games*
* ***University of York, United Kingdom ,The changing face of desktop video game monetisation: An exploration of exposure to loot boxes, pay to win, and cosmetic microtransactions in the***

***Most-played Steam games of 2010-2019. (2020, May 7).*** *Retrieved from https://pmc.ncbi.nlm.nih.gov/articles/PMC7205278/*

* ***Nazir S. Hawi et al., Journal of Behavioral Addictions. "Internet Gaming Disorder in Lebanon: Relationships with Age, Sleep Habits, and Academic Achievement."(2018,Feb).*** *Retrieved from*

*https://*[*www.researchgate.net/publication/323456560\_Internet\_gaming\_disorder\_in\_Lebanon\_*](http://www.researchgate.net/publication/323456560_Internet_gaming_disorder_in_Lebanon_) *Relationships\_with\_age\_sleep\_habits\_and\_academic\_achievement*

# Requirements:

## Use Case Diagram – “Road Bound Legend” - Game Application

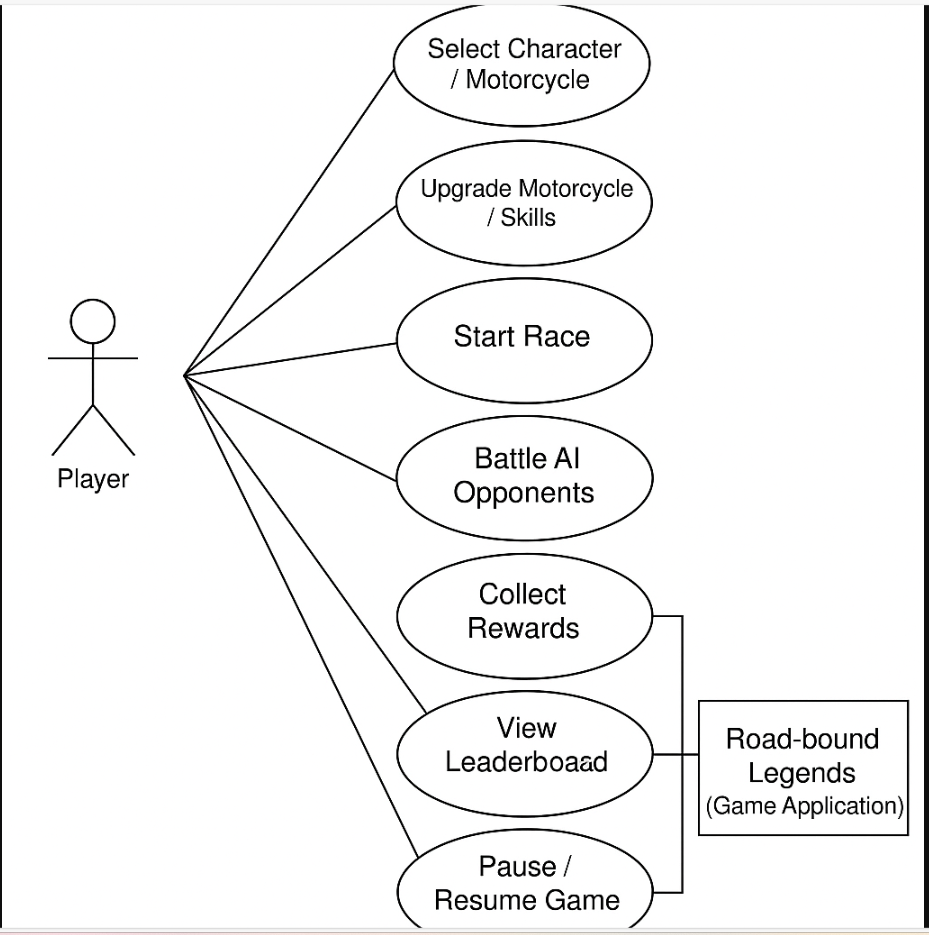
The use case diagram illustrates the interaction between the actors and subsystems involved in the game application. The diagram identifies two key actors: the "Player" and the "Game Application." The "Player" interacts with the application by performing various actions such as selecting characters, upgrading skills, battling enemies, collecting rewards, and progressing through game levels.

The “Road Bound Legend Game Application” acts as the central component and facilitates the interaction between the player and the subsystems. It provides an engaging interface and coordinates the functionalities offered by the subsystems to ensure an immersive gaming experience.

The subsystems identified in the diagram include:

1. **Character Selection & Upgrade Subsystem:** This subsystem allows players to select characters, modify their appearances, and enhance skills or powers with game-earned points or rewards.
2. **Battle & Enemy Interaction Subsystem:** Players can fight different bosses and creatures in different environments or levels because to this subsystem. It generates dynamic combat experiences by managing attack, defence, and strategy mechanisms.
3. **Reward Collection Subsystem:** Players can receive rewards from this subsystem upon finishing missions, winning fights, or hitting milestones. Coins, weapons, upgrades, and special items are some of the possible rewards.
4. **Game Progression & Level Management Subsystem:** Throughout the game, this subsystem controls the player's journey through different stages or levels. It unlocks new levels, keeps track of the player's progress, and modifies difficulty according to performance.
5. **User Interface Subsystem:** The user interface subsystem guarantees that players have an interactive, appealing, and easy-to-use experience. It improves the gaming experience by including menus, in-game controls, notifications, and visual effects.

The "Road of Legend" gaming application's use case diagram shows the player's interactions and the flow of actions between the various subsystems. "Road of Legend" provides players with an exciting and exciting gaming experience by using dynamic battle systems, interacting interfaces, and game design concepts. The software allows players to explore, battle, upgrade, and overcome obstacles as they go through their legendary journey.



*Figure 1 – Use Case Diagram 1*

## Functional Requirements:

**Motorcycle Customization & Selection:**

* Before the game begins, the player will have the option to choose from a variety of motorcycles.
* Each motorcycle shall offer a unique gaming experience based on performance, speed, handling, and control.
* Motorcycles can be upgraded or customized through the system by using achievements or awards that have been obtained.

**Game Modes & Gameplay Control:**  
Using keyboard, touch, or controller inputs, players will be able to operate their motorcycle in the game.  
The system shall support different game modes:

* Time Trial Mode
* Free-Ride Mode
* Racing mode with AI opponents.

While playing the game, the system will enable players to pause, resume, and stop.

**Collision Detection & Physics Simulation:**

* To provide a realistic driving experience, the system will use physics-based motorcycle movement.
* The system will identify collisions between motorcyclists, obstacles, and objects in the surrounding area.
* The biker character will use ragdoll physics to recreate crashes and falls realistically.

**Reward Collection & Upgrades:**

* The player shall receive rewards (coins, items, motorcycle parts) after completing races, challenges, or defeating AI opponents.
* Rewards shall be used to unlock or upgrade motorcycles and customize gameplay.

**Interaction of the User Interface:**

* For every device, the system must offer an engaging and attractive user interface.
* It will be easy for the player to monitor notifications, browse menus, and access settings.

**The system will show gaming details such as:**

* Speed
* Timer
* Position
* Minimap
* Player Information
* A high-scoring table or leaderboard

## Data Requirements:

* Player profile data (username, score, achievements).
* Game state and progress data (levels completed, bikes unlocked).
* Motorcycle specifications and performance metrics.
* Track/map data (terrain type, obstacles, checkpoints).
* Leaderboard data (top scores, global and friends’ scores).

## Non-Functional Requirements:

**Performance:**

* Keep your frame rate at 60 FPS or higher for fluid gameplay.
* Less than 100 milliseconds is the input response time.
* Handle at least 5 AI motorcycles without noticeable lag.

**Dependability**:

* The game must not crash while playing normally.
* The game should automatically save its progress every two minutes or after every race.
* Game progress data should not get corrupted on unexpected shutdown.

**Maintainability and Supportability**:

* Modular game code structure for easy updates (adding new maps, bikes).
* Well-documented code for developers.
* Integrated logging for error tracking.
* Easy addition of new AI behaviors or motorcycles.

**Security:**

* Prevent cheating or hacking by manipulating input.
* Prevent unauthorized alteration of saved files.
* If new features are added to the internet, HTTPS will be used to transmit data securely.

**Usability and Humanity**:

* An easy-to-use interface.
* Instructions, labels, and icons are all clear.
* Controls can be customized according to the user's preferences.
* easy-to-follow instructions.

**Look and Feel:**

* The game should have a realistic and immersive design style.
* Motorcycle models should be detailed with realistic textures and animations.
* Background music and sound effects should enhance the experience without being distracting

**Operational and Environmental:**

* The game should work in various lighting conditions.
* Should support both online and offline modes.

**Cultural and Political:**

* No politically sensitive or offensive material.
* Localized language support for multiple regions.
* Culturally sensitive symbols or content must be avoided.

**Legal:**

* The game's music, graphics, and other materials must comply by copyright regulations.
* Respect data privacy laws (e.g., COPPA, GDPR).
* Include appropriate licensing for third-party engines and libraries.

## Design:

## System Architecture:

The following layers create "Road-bound Legends"' modular separation of concerns architecture:  
Player interactions, menus, HUDs, and notifications are handled by the user interface layer.  
Race mechanics, AI behaviours, collision detection, scoring, and reward systems are all managed by the game logic layer.  
Physics Engine Layer: Applies Unity's physics for realistic motorcycle handling and collision responses.  
Data Management Layer: Handles saving and loading player profiles, achievements, and leaderboards.

## User Interface:

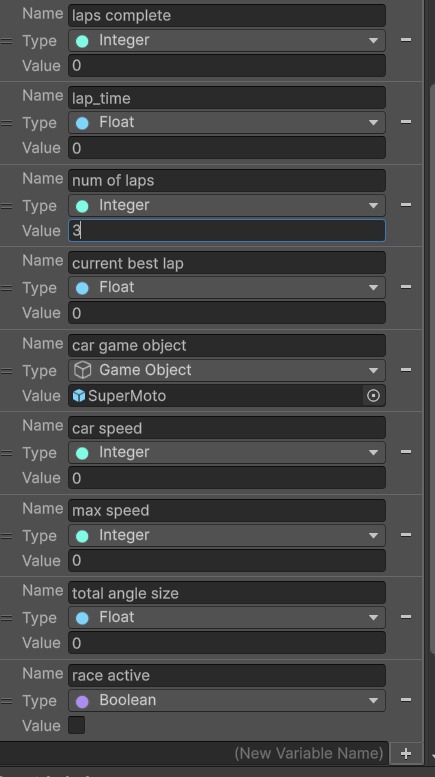
* + Dynamic HUD showing speed, time, and position.
  + Countdown time before race starts.
  + Lap time tracker.
  + Race start and finish prompt.
  + Menus for starting, pausing, and ending races.
  + Garage/Customization screen for bikes.
  + Leaderboard screen.

## Class Table:

|  |  |
| --- | --- |
| **Class Name** | **Description** |
| PlayerProfile | Stores player name, achievements, and game progress. |
| Motorcycle | Represents a motorcycle, including stats like speed, acceleration, handling. |
| RaceManager | Handles race start, lap counting, timer, and AI management. |
| AI\_Opponent | Controls AI motorcycle behavior using reinforcement learning or predefined paths. |
| RewardManager | Controls collection of rewards and upgrades. |
| CountdownManger | Controls countdown before race start, sets raceActive flag. |
| Laptimer | Tracks lap completion times, calculates best lap. |

## Variable Table:

|  |  |  |
| --- | --- | --- |
| **Variable Name** | **Type** | **Description** |
| Laps\_complete | Integer | Tracks how many laps the player has finished during the race. |
| Lap\_time | Float | Stores the duration of the player’s last completed lap. |
| Current\_best\_lap | Float | Holds the best (lowest) lap time achieved in the race. |
| Num\_of\_laps | Integer | Defines the total number of laps required to finish the race. |
| Race\_active | Boolean | Indicates whether the race is currently running(true) or not (false). |
| Car\_game\_object | GameObject | Reference to the motorcycle or vehicle controlled by the player. |
| Car\_speed | Integer | Tracks the real-time speed of the player’s vehicle. |
| Max\_speed | Integer | Defines the maximum allowed speed of the vehicle. |
| Total\_angle\_size | Float | Measures the total rotation angle for steering the motorcycle. |



## UI WireFrames:

* + Main menu:



* + Game HUD( lapcounter, timer):



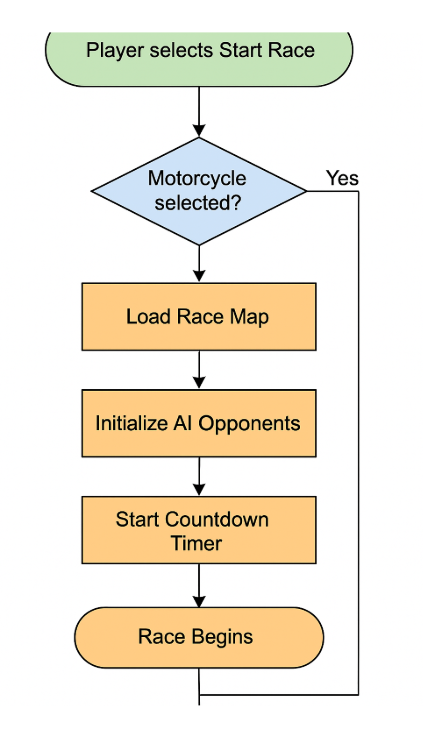
* + Garage(bike selection):



* + End Race Screen:



## Activity Diagram:

**if statetment eza hatyna backspace bytlaa mnel game**

## Tools And Technologies:

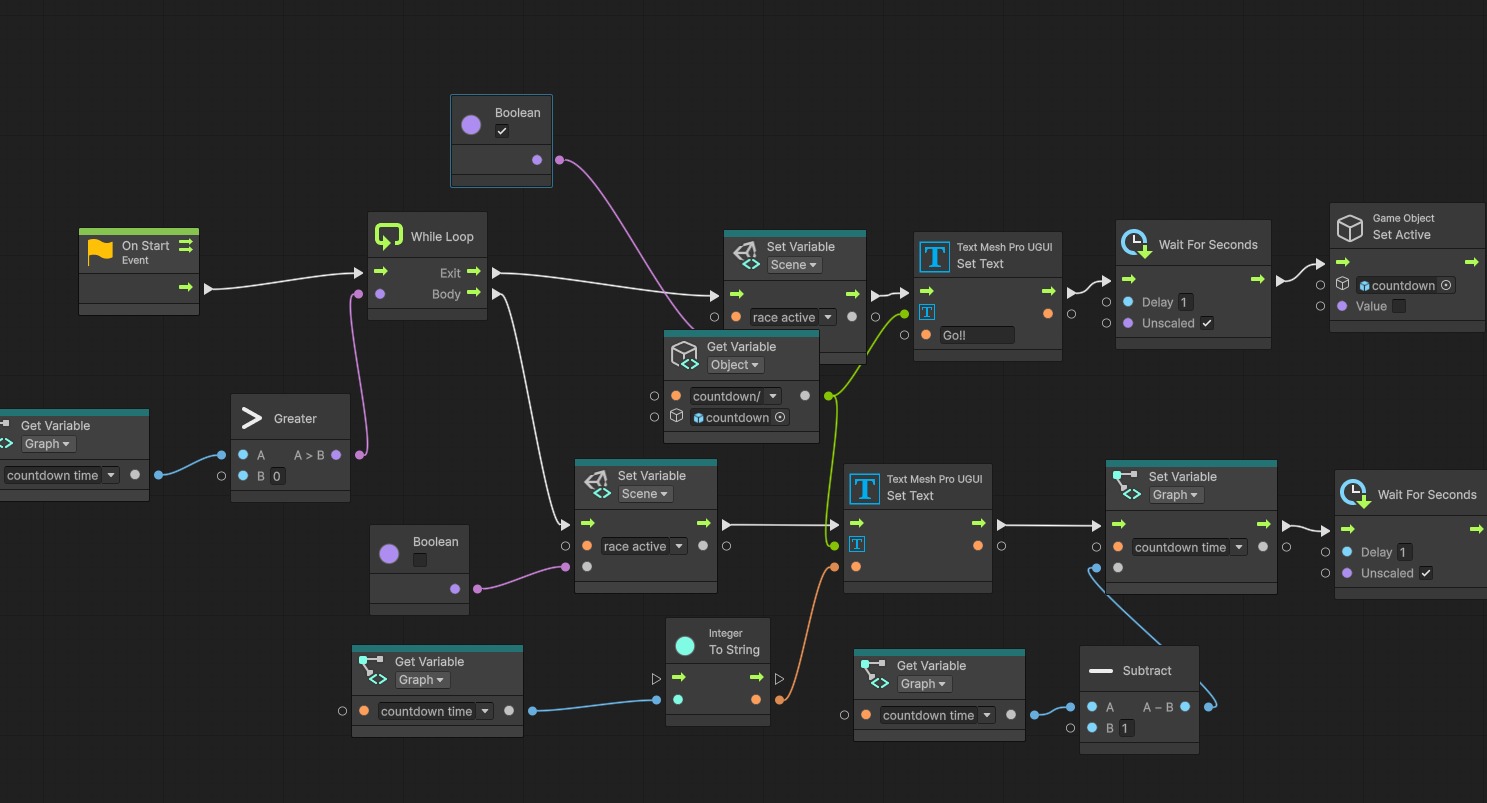
|  |  |
| --- | --- |
| **Tool** | **Purpose** |
| Unity 6 | Game engine |
| Blender | 3D modeling for motorcycles, environments |
| Git | Version control |
| Visual Studio | C# scripting for game logic |
| Animation Rigging (unity package) | Used for procedural biker animations and inverse kinematics. |
| Unity Asset Store | Asset acquisition (maps, characters, effects) |

## Coding Standards:

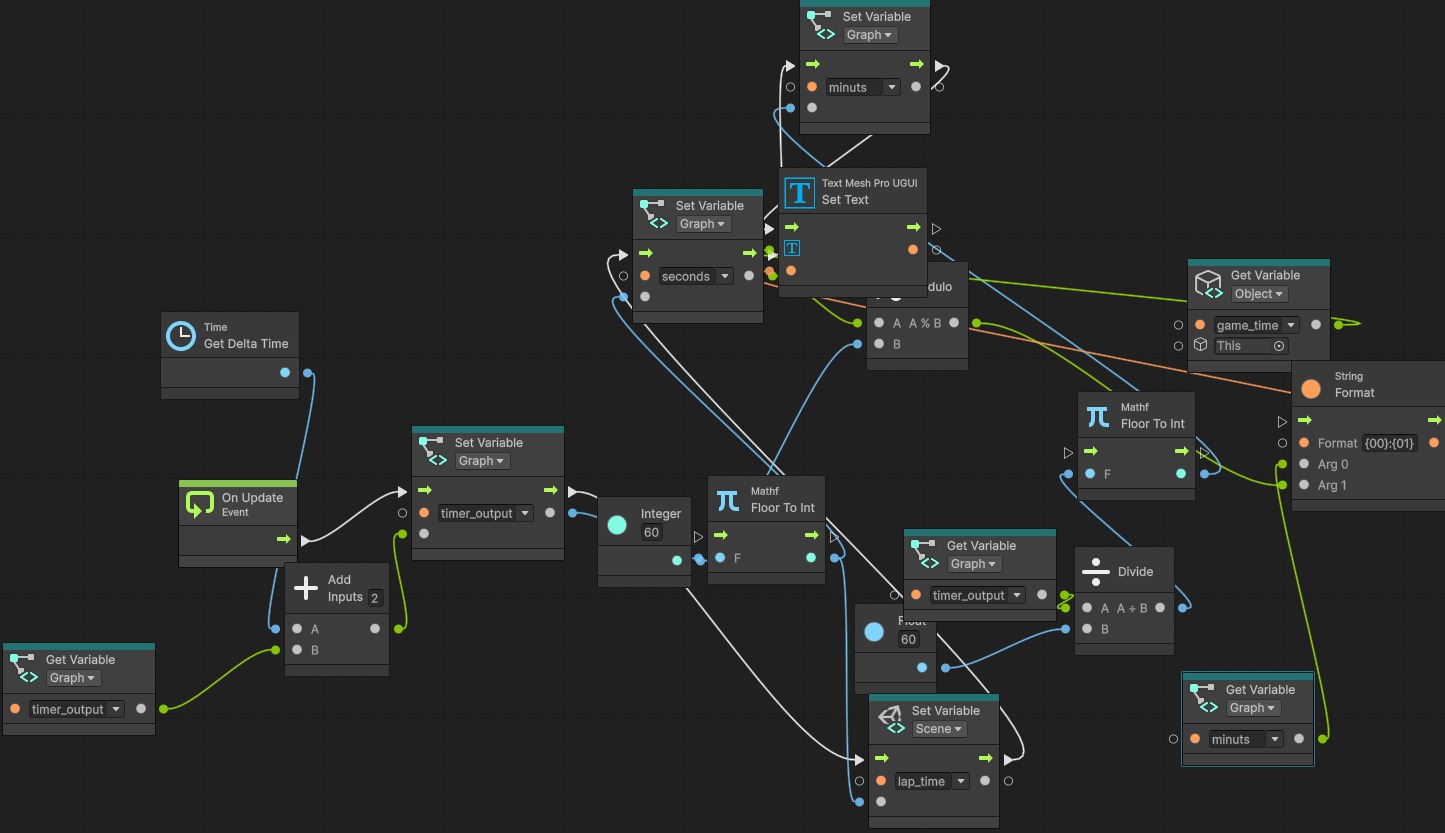
* 1. Naming Conventions:
     + Classes and Scripts: Use PascalCase (RaceManager,Laptimer)
     + Variables: Use camelCase (lapsComplete,raceActive)
     + Constants: Use all UPPERCASE\_WITH\_UNDERSCORES(MAX\_SPEED)
     + Graph Elements(for Visual Scripting Variables):
       - Scene Variables: Use descriptive names like lapsComplete,raceActive.
       - Object Variables: Match object roles like carGameobject, countdownText.
  2. Script and Graph Organization:
     + Each script should have one main responsibility.
     + Use Comments in C# script and sticky notes inside visual scripting graphs to explain complex flows.
  3. Modular Code Structure:
     + Break down systems into separate managers:
       - Countdown Manager
       - Lap Timer
       - Race Manager
  4. Performance Optimization:
     + Minimize Update() Loops
     + Use Unscaled Time ( WaitForSeconds Unscaled)
  5. Error Handling and Safe Practice:
     + Always check if a GameObject is active
     + Use try-catch blocks if any external plugins

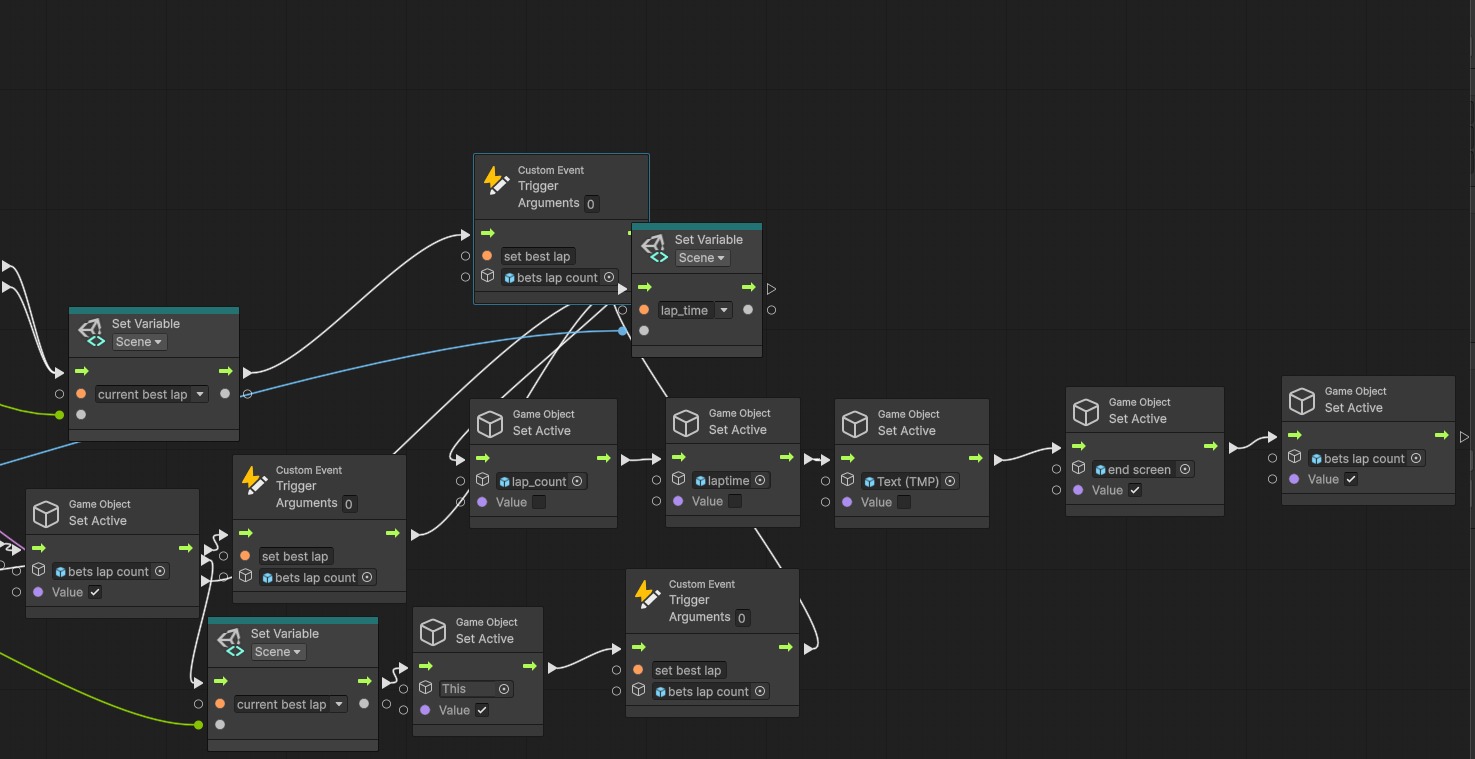
## Implementation:

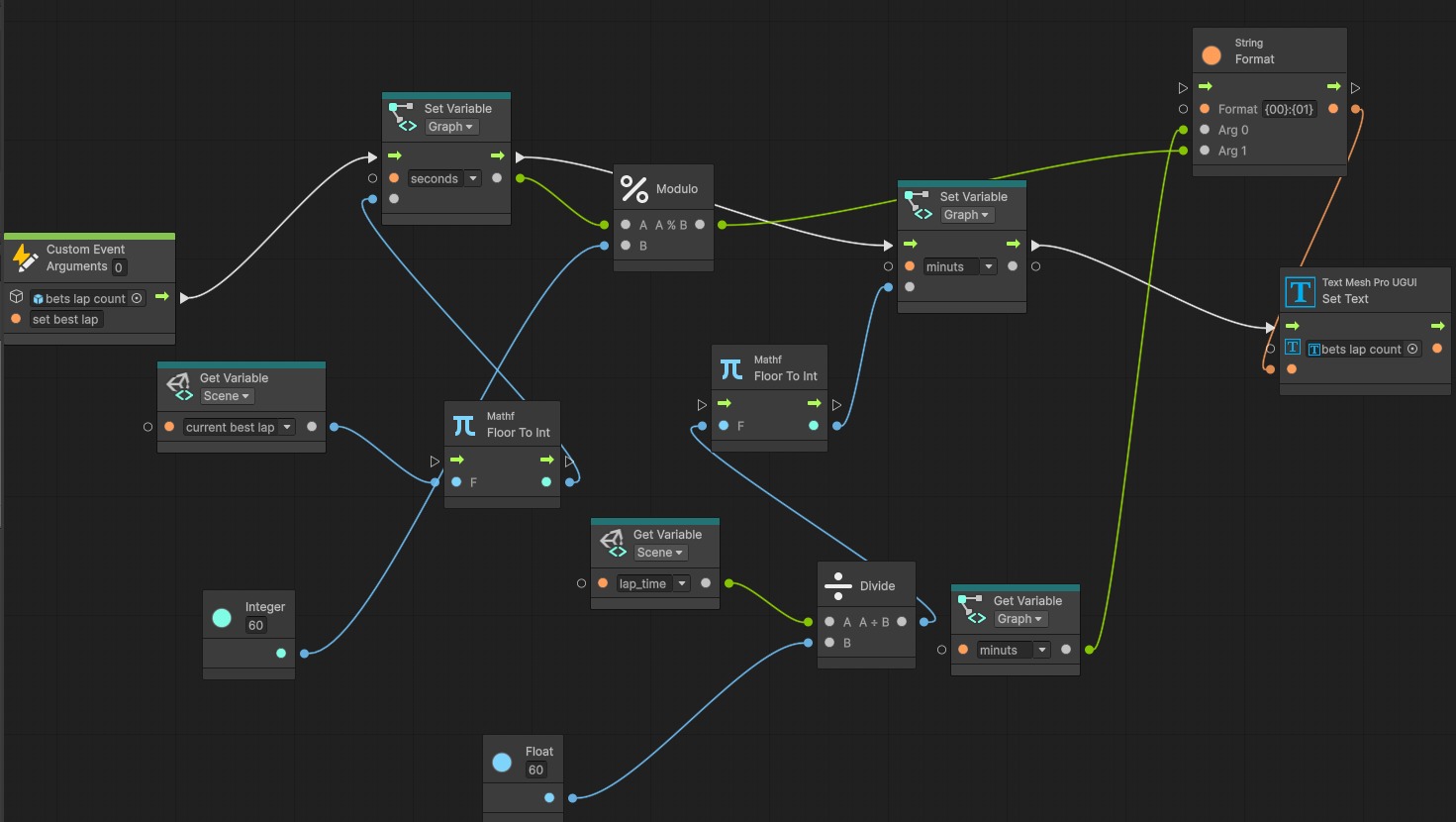
* **Countdown System:**
  + While Loop runs and updates the countdown timer every second.
  + TextMeshPro updates the displayed countdown numbers.
  + When countdown reaches 0, it sets race\_active = true.
  + "Go!" text is shown to the player to start the race.



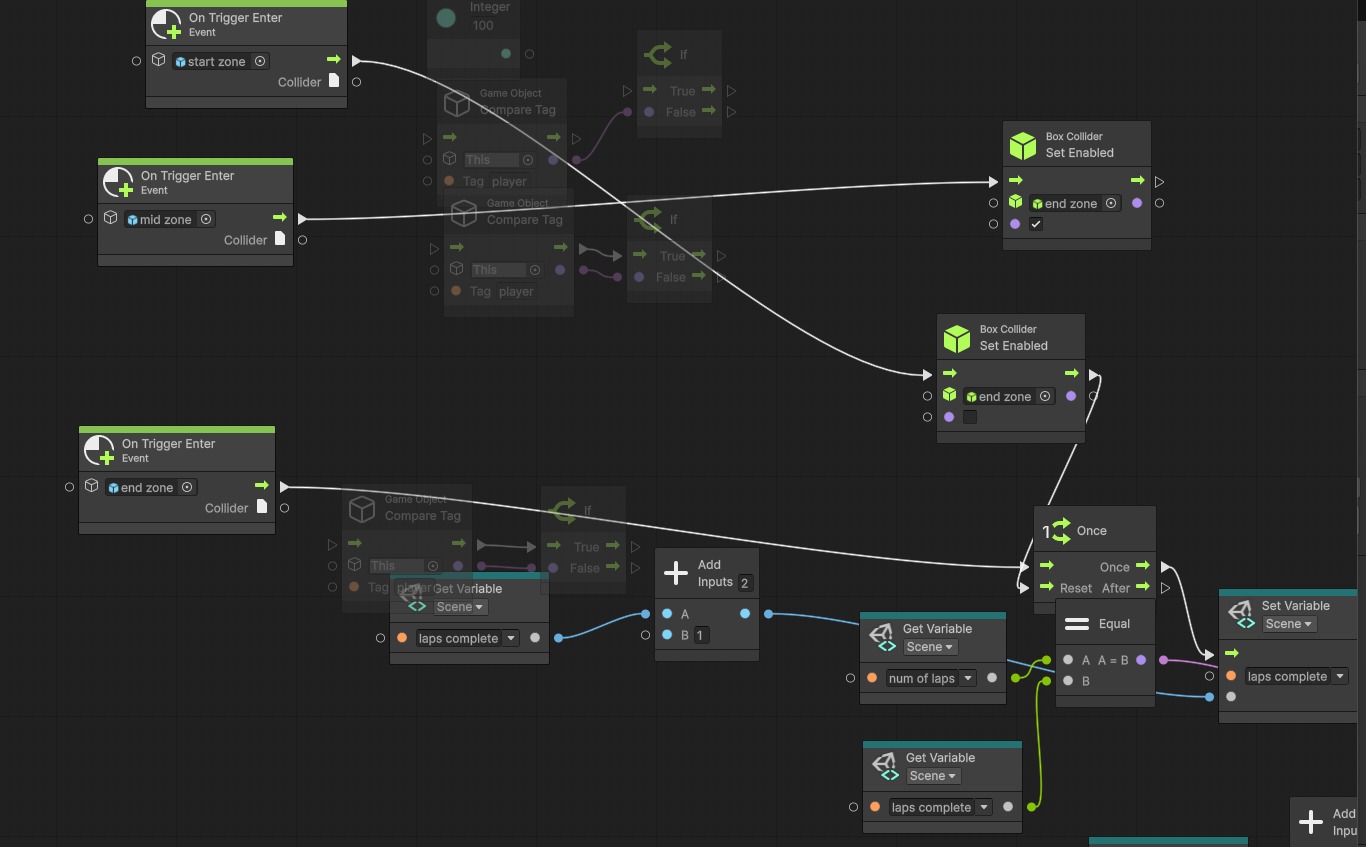
* **Lap Timer System:**
  + uses WaitForSeconds Unscaled to track in real-time, increments the timer\_output variable.
  + uses Mathf.Floor and modulo operations to convert total seconds into minutes and second.
  + uses TextMeshPro to update the display with the current lap time.

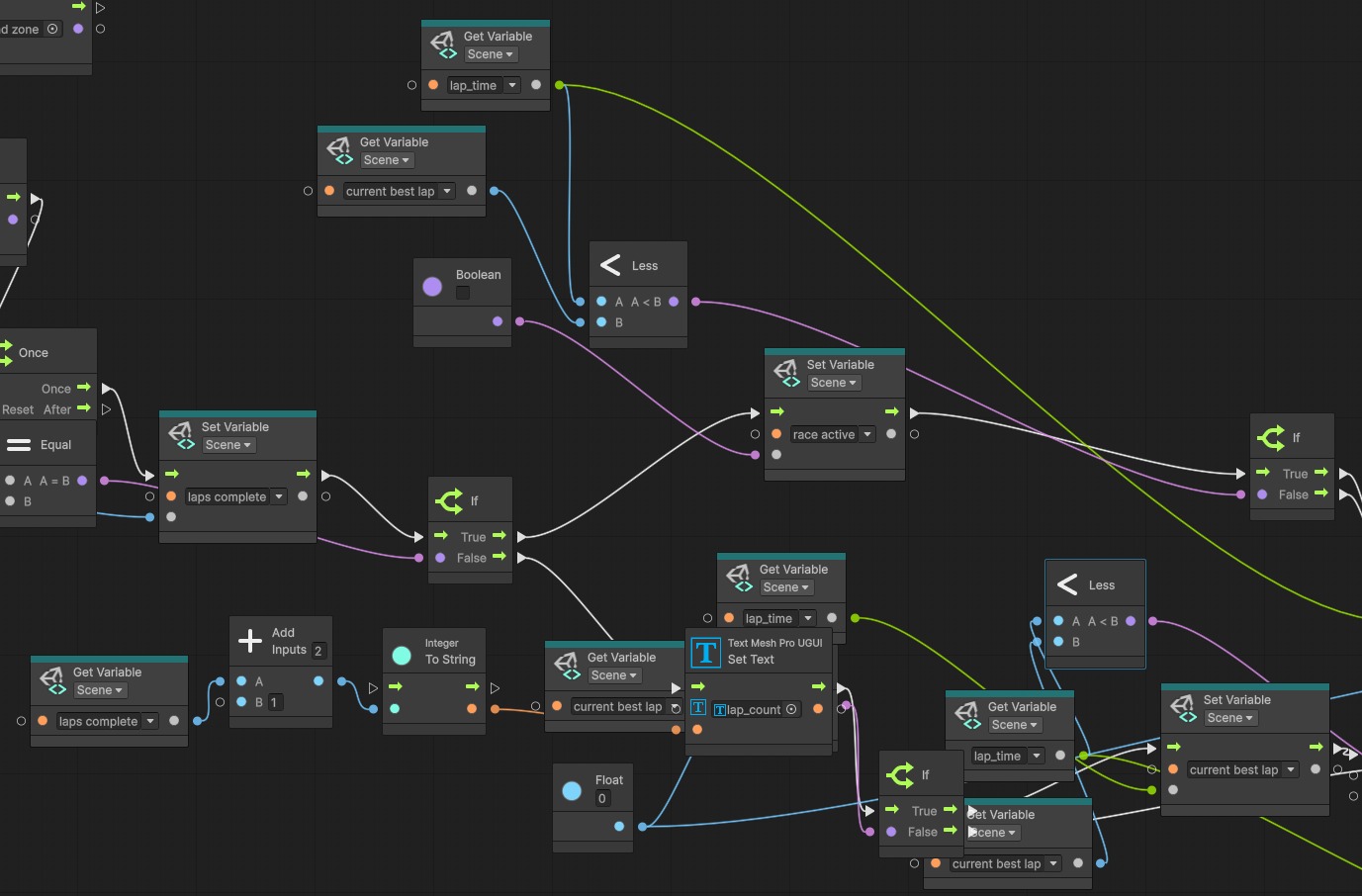






* **Race Manager System:**
  + OnTriggerEnter detects when the player crosses:
  + Race start is set by Start Zone.
  + Checkpoints in the mid-zone are optional.
  + End Zone → Finishes one lap.
  + After finishing a lap:
    - Increments laps\_complete.
    - updated if lap\_time is better than current\_best\_lap.
  + The End Screen is displayed and the race is over if laps\_complete == num\_of\_laps.





* **Physics System**:
  + Simulates realistic motorcycle behavior using the Simple Motocross Physics package:
  + Joint-equipped suspension system.
  + IK rider animations that are procedural.
  + Completely compatible with:
  + VR, iOS, Android, and PC platforms.
  + Supports rendering pipelines that are Standard, URP, and HDRP.
* **UI System**:
  + UI System HUD elements such as lap counter, and timers dynamically updated during gameplay.
  + Countdown A timer is displayed at the beginning of the race.
  + Best Lap Time updated after each lap.
  + End Screen displays player performance and race summary.
* **AI Implementation in AnyCarAI**
* The AI system in AnyCarAI is based on a waypoint-following mechanism combined with obstacle detection using sensor-based logic. The core script, embedded in the AIScript object within the prefab, allows cars to autonomously navigate a scene in Unity using a predefined path and dynamic behavior controls. Here's how it works:
* **Pathfinding via Waypoints:**
* The AI uses a series of waypoint nodes (set up using the WaypointsPath prefab) that define the track the car should follow. The script interpolates between these waypoints to generate a smooth path or direct transitions, depending on user settings.
* **Sensors and Obstacle Detection:**
  + The AI car is equipped with adjustable virtual sensors (configured through angle and distance parameters) that detect nearby objects. When an obstacle is detected within a defined range, the AI either brakes or calculates an alternative maneuver like reversing.
* **Human-like Driving Simulation:**
* The AI features a "Humanizator" module that introduces acceleration, braking, and steering sensitivity to mimic human behavior. This creates more natural driving motion rather than rigid, robotic navigation.
* **Pursuit Mode:**
* An optional pursuit AI allows the car to track and follow another game object dynamically, useful in chase or escort scenarios.
* **Wheel and Physics Management:**
  + The script adapts to any car model by referencing wheel colliders and adjusting physics-based parameters like suspension, friction, and engine torque, ensuring realistic driving responses.
* **Audio and Visual Effects:**
  + The AI integrates with Unity’s audio mixer and particle systems to simulate real-time car effects like engine noise, exhaust, skidding, and collisions, enhancing realism.
* **Damage System (Optional):**
  + If enabled, the mesh of the car deforms on impact, and parts can break off based on collision intensity, managed through a customizable array.

## Testing

* **Unit Testing:**
* CountdownManager:
  + Every second, the countdown is confirmed to decrease accurately.
  + As soon as you reach zero, the race begins automatically.
* LapTimer:
  + Every frame is precisely updated by the verified timer.
  + Displays the correct conversion from seconds to minutes/seconds.
* RaceManager:
  + examined the rationale of lap counting.
  + Following each lap, the best lap is appropriately recorded.
* **Integration Testing:**
  + Tested combined CountdownManager, LapTimer, and RaceManager:
    - Following the countdown, the race begins.
    - A timer is on during the race.
    - The race ends when all of the required laps are completed.
* **Performance Testing:**
  + Steady 60 frames per second on mid-range PCs.
  + Handled five AI opponents without seeing any frame dips.
* **User Testing**:
  + Players provided positive feedback:
    - Countdown clarity.
    - Lap recording and fluid control.
    - No UI changes or lags between laps.
* **Bug Tracking:**
  + Bugs are reported using git issues:
    - The countdown is now accurately shown.
    - After pausing, the lap timer freezes(fixed).
    - When passing the mid-zone rather than the end-zone, the race ends incorrectly(fixed).