VIETNAM NATIONAL UNIVERSITY, HO CHI MINH CITY
HO CHI MINH CITY UNIVERSITY OF TECHNOLOGY
FACULTY OF COMPUTER SCIENCE AND ENGINEERING

GAME PROGRAMMING (CO3045) - SEMESTER 241

Assignment Report

FRUIT NINJA

Lecturer: Phan Trần Minh Khuê - **Class**: CC01

No.	Name	Student ID	Email address
1	Hồng Huy Mẫn	2052593	man.hong9102@hcmut.edu.vn
2	Hồ Mạnh Quân	1952941	quan.hobachkhoa@hcmut.edu.vn
3	Nguyễn Anh Tuấn	1953070	tuan.nguyen62@hcmut.edu.vn
4	La Kỳ Phương	2014203	phuong.la0120@hcmut.edu.vn

Ho Chi Minh City, November 2024

Contents

List of Figures

1 Contributions

No.	Name	Student ID	Contribution	Percentage
1	Hồng Huy Mẫn	2052593	Game developer, Report	100%
2	Hồ Mạnh Quân	1952941	Material Research, Report, Leader	100%
3	Nguyễn Anh Tuấn	1953070	Game developer, Report	100%
4	La Kỳ Phương	2014203	Material Research, Report	100%

2 Introduction

Storyline and Target Market:

In today's fast-paced world, especially in developing and developed countries, many working professionals find themselves with limited time to enjoy games that require long play sessions. With hectic schedules and short breaks, there's a growing need for games that offer quick and satisfying entertainment.

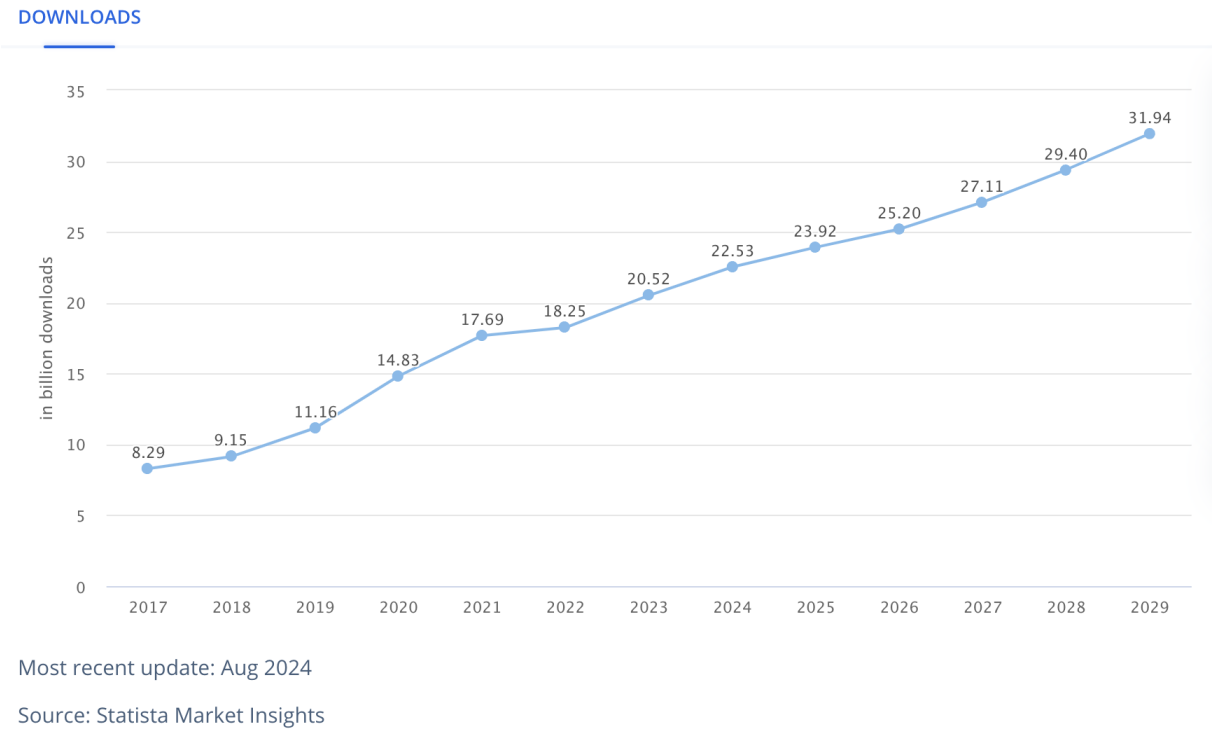

Figure 1: *Number of downloads casual game increasing every year*

That's why we created Ninja Fruit—a game designed to help you unwind and have fun in just a few minutes. Whether you're taking a quick break at work or relaxing during your lunch hour, Ninja Fruit provides the perfect escape without demanding too much of your time. Plus, it helps you sharpen important skills like concentration and quick reflexes, all while enjoying some tasty virtual fruit slicing!

Game Mechanics:

Fruit Ninja (queue stimulation version) delivers an exciting and fast-paced gaming experience that emphasizes quick reflexes and precise movements. What makes our game unique is the innovative queue system we've incorporated into the gameplay. When you start the game, fruits appear on the screen and line up in a queue. Only the fruit at the front of the queue can be sliced to earn points. If you try to slice a fruit that's not first in line, nothing happens. This system keeps the game straightforward but adds an extra layer of challenge, ensuring you stay focused and react swiftly to slice the right fruit at the right time.

Rules:

1. ***Slice in the Right Order:*** Only the fruit at the front of the queue can be sliced. Successfully slicing it removes it from the queue and awards you points.

Figure 2: *Queue in real life*

2. ***Wrong Order Slices:*** Trying to slice a fruit that's not at the front won't do anything, keeping you on track to follow the correct sequence.

3. ***Missed Fruits:*** If the fruit at the front of the queue falls off the screen without being sliced, the game ends.

4. ***Avoid the Bombs:*** Occasionally, bombs will join the queue. Slicing a bomb will end the game immediately, adding an element of danger and requiring careful attention.

Objectives:
The main goal of Fruit Ninja is to slice as many fruits as possible in the correct order to rack up a high score while avoiding bombs to prevent the game from ending. This not only provides a fun and engaging experience but also helps you develop your observation skills, concentration, and quick reflexes. The game's scoring system and ongoing challenges encourage you to keep improving, giving you a rewarding sense of achievement with every slice.

Platform Compatibility:
We developed Fruit Ninja to be playable on both laptops and desktops, ensuring that you can enjoy the game on a variety of devices. The user-friendly interface is optimized for different screen sizes, whether you're playing on a large desktop monitor or a compact laptop screen. You can easily control the blade by clicking and dragging with your mouse on Windows, swiping with three fingers on macOS, or using touch gestures on touchscreen laptops. This flexibility makes the game accessible and convenient, no matter where you are or what device you're using. Furthermore, due to the enormous size of gaming market on PC, we decided to make game on this platform

In conclusion, with its innovative queue-based design, Ninja Fruit offers a fresh twist on traditional fruit-slicing games. It combines simple mechanics with engaging challenges, making it perfect for players who want quick, enjoyable gameplay that fits into their busy lives. Whether you're looking to relieve stress or just have a bit of fun, Ninja Fruit provides a delightful and accessible experience for everyone.

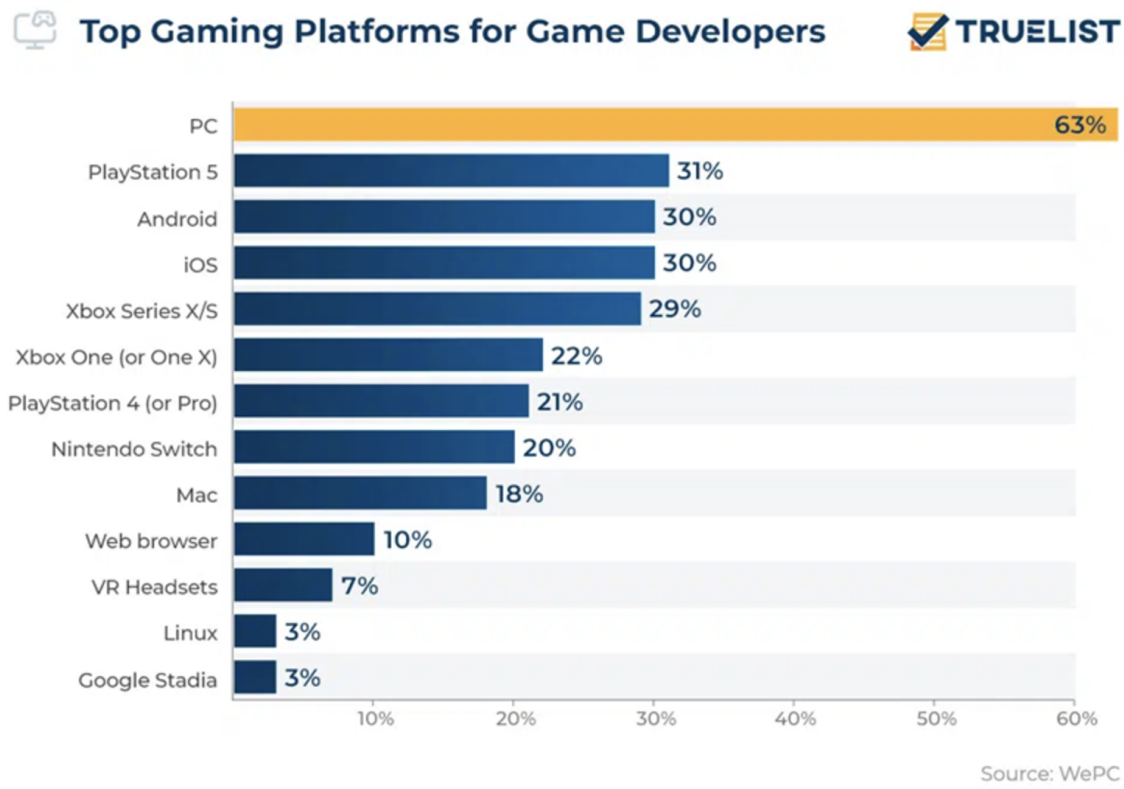

Figure 3: *Top gaming platform for developers*

3 Analysis and Design

3.1 Conceptual Framework

We re-designed the Ninja Fruit game using a queue system to slightly change the traditional fruit-slicing experience. In this game, whenever a new piece of fruit appears, it goes to the end of a line (a queue). Only the fruit at the very front of this line can be sliced for points. If you try to slice a fruit that isn't at the front yet, nothing happens. This forces you to watch carefully instead of just relying on quick reactions.

Because you must slice in the correct order, you face not only time pressure but also the challenge of identifying the right target. The front fruit could fall off the screen at any moment, and if it does so before you manage to slice it, you lose immediately. This rule keeps you focused. You must constantly pay attention to the order in which fruits appear and can't just swipe wildly at everything you see.

Due to this change, players can't simply slice any fruit that shows up. Instead, they have to truly notice which fruit comes first and follow the sequence. This helps improve careful observation, calmness, and the ability to size up each situation. You become more active in your decision-making, not just fast with your hands and eyes. You learn which fruit you need to slice next and the perfect moment to do it, so you don't miss your chance.

By using the queue system, we want to create a new challenge—one where winning depends not just on speed, but also on staying focused and making the right decisions at the right time. It's not just a different way of playing, but also a chance for players to sharpen their observation and

concentration—two important skills we often overlook when we're just trying to relax and have fun.

Below is the class diagram showing the structure of the whole project and the relationships between objects. The diagram includes five main classes: **Blade**, **Bomb**, **Fruit**, **GameManager**, and **Spawner**.

The **Blade** class is responsible for managing the fruit-slicing mechanics. It handles player input events to determine when to start, continue, and stop slicing.
Key attributes:

- `sliceForce`: The force applied when slicing.
- `minSliceVelocity`: The minimum velocity required to recognize a valid slice.
- `mainCamera, sliceCollider, sliceTrail`: Components necessary for calculating and displaying the slicing action.

Key methods:

- `Awake(), OnEnable(), OnDisable(), Update()`: Lifecycle methods for initializing and managing the slicing state.
- `StartSlice(), StopSlice(), ContinueSlice()`: Handle different stages of the slicing process.

The **Bomb** class represents the bombs in the game. When a bomb collides with the Blade, it triggers the game-over event.
Key method:

- `OnTriggerEnter(Collider other)`: Detects collision with the **Blade** and calls the `Explode` method from **GameManager**.

The **Fruit** class manages the fruit objects in the game, including their intact and sliced states.
Key attributes:

- `whole, sliced`: Objects representing the fruit's intact and sliced states.
- `fruitRigidbody, fruitCollider, juiceEffect`: Physical components and effects when the fruit is sliced.
- `points`: Points awarded when the fruit is successfully sliced.
- `isSliced`: A flag indicating whether the fruit has been sliced.

Key methods:

- `Slice(Vector3 direction, Vector3 position, float force)`: Handles the slicing process, including changing states, creating effects, and updating the score.
- `OnTriggerEnter(Collider other)`: Detects collision with the **Blade** to perform slicing.
- `Update()`: Monitors the fruit's position to determine if it falls below the screen, triggering game over.

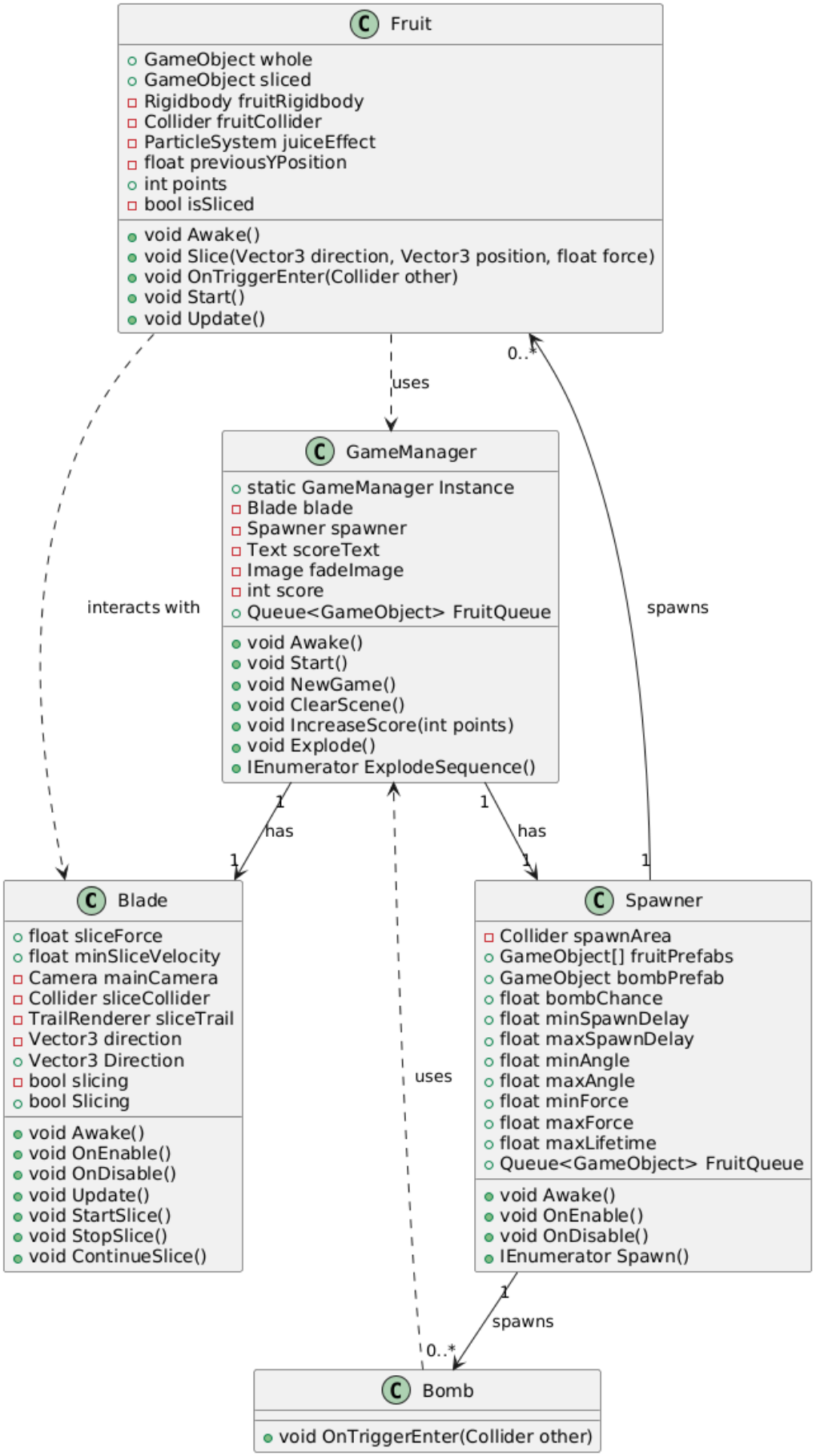

Figure 4: *Conceptual class diagram*

GameManager is the main management class of the game, utilizing the Singleton pattern to ensure only one instance exists throughout the game.
Key attributes:

- `blade, spawner`: Components managing **Blade** and **Spawner**.
- `scoreText, fadeImage`: UI elements displaying the score and transition effects.
- `score`: The player's current score.
- `FruitQueue`: A queue managing the order in which fruits appear.

Key methods:

- `Awake()`: Sets up the Singleton instance.
- `Start(), NewGame()`: Initialize and start a new game.
- `ClearScene()`: Removes existing fruit and bomb objects from the scene.
- `IncreaseScore(int points)`: Increases the score and updates the UI.
- `Explode(), ExplodeSequence()`: Handles the game-over sequence with fade effects and restarts the game.

The **Spawner** class is responsible for creating Fruit and Bomb objects in the game at random positions within the spawn area.
Key attributes:

- `fruitPrefabs, bombPrefab`: Prefabs for spawning fruits and bombs.
- `bombChance`: The probability of spawning a bomb instead of a fruit.
- `minSpawnDelay, maxSpawnDelay`: Time intervals between spawns.
- `minAngle, maxAngle`: Random rotation angles for spawned objects.
- `minForce, maxForce`: Force applied when spawning objects.
- `maxLifetime`: Maximum lifetime of spawned objects before they are destroyed.
- `FruitQueue`: A queue managing the order in which fruits are spawned.

Key methods:

- `Awake()`: Initializes the spawn queue.
- `Spawn()`: Coroutine responsible for creating objects at specified intervals and conditions.

3.2 Objects

In this Ninja Fruit game, players will encounter six main objects: Apple, Bomb, Kiwi, Lemon, Orange, and Watermelon. Each fruit type has its unique shape and color. Diversifying the types of fruits with different visual appeal give the game attractive look. Players must pay attention to the order in which fruits appear to slice them correctly, thereby honing their observation skills, concentration, and quick reflexes. Additionally, the presence of bombs introduces an element of danger, requiring players to remain vigilant and strategic with each slicing action.

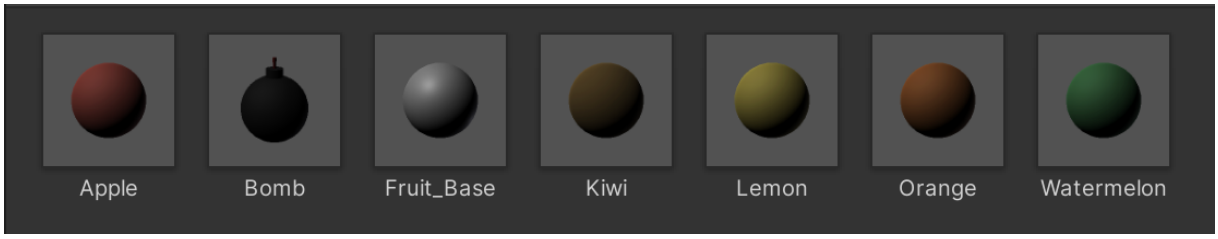

Figure 5: *Objects of the game*

3.3 Game Mechanics

The gameplay revolves around a unique queue-based system. When the game starts, fruits begin to appear on the screen and are added to the end of a queue. This queue dictates the order in which fruits must be sliced. Players can only successfully slice the fruit that is at the front of the queue. Attempting to slice any other fruit that is not first in line has no effect, ensuring that players must follow the correct sequence. Also, in order to make this game more interesting, we add some bombs - appear by small chance. If players do not observe and instantly recognize, the players will lose, too.

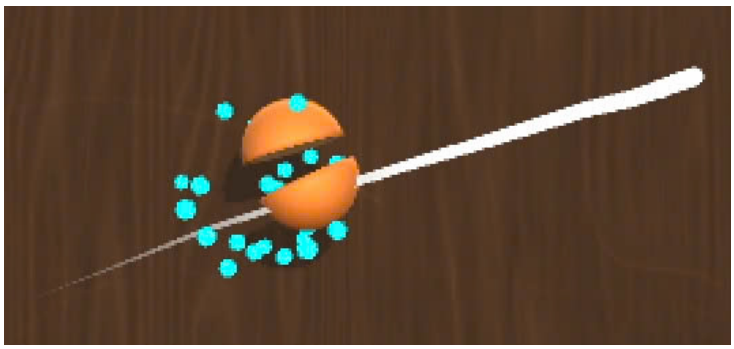

Figure 7: *The effects appear when slicing the correct ordered fruit.*

The **Blade** is the player's tool for slicing fruits. It responds to player inputs, allowing them to slice the front-most fruit in the queue. When a player slices the correct fruit, it is removed from the queue, and the player's score increases. However, if a fruit at the front of the queue falls off the screen without being sliced, the game ends immediately. This rule adds a layer of challenge, as players must stay focused and react quickly to prevent any fruit from escaping.

The **Spawner** continuously generates new fruits, adding them to the queue. It occasionally generates bombs which is not added to the Queue. **Bombs** introduce additional difficulty by triggering a game-over event if they collide with the **Blade**. This keeps the gameplay dynamic and requires players to maintain high levels of concentration and precision. The **GameManager** oversees the entire game

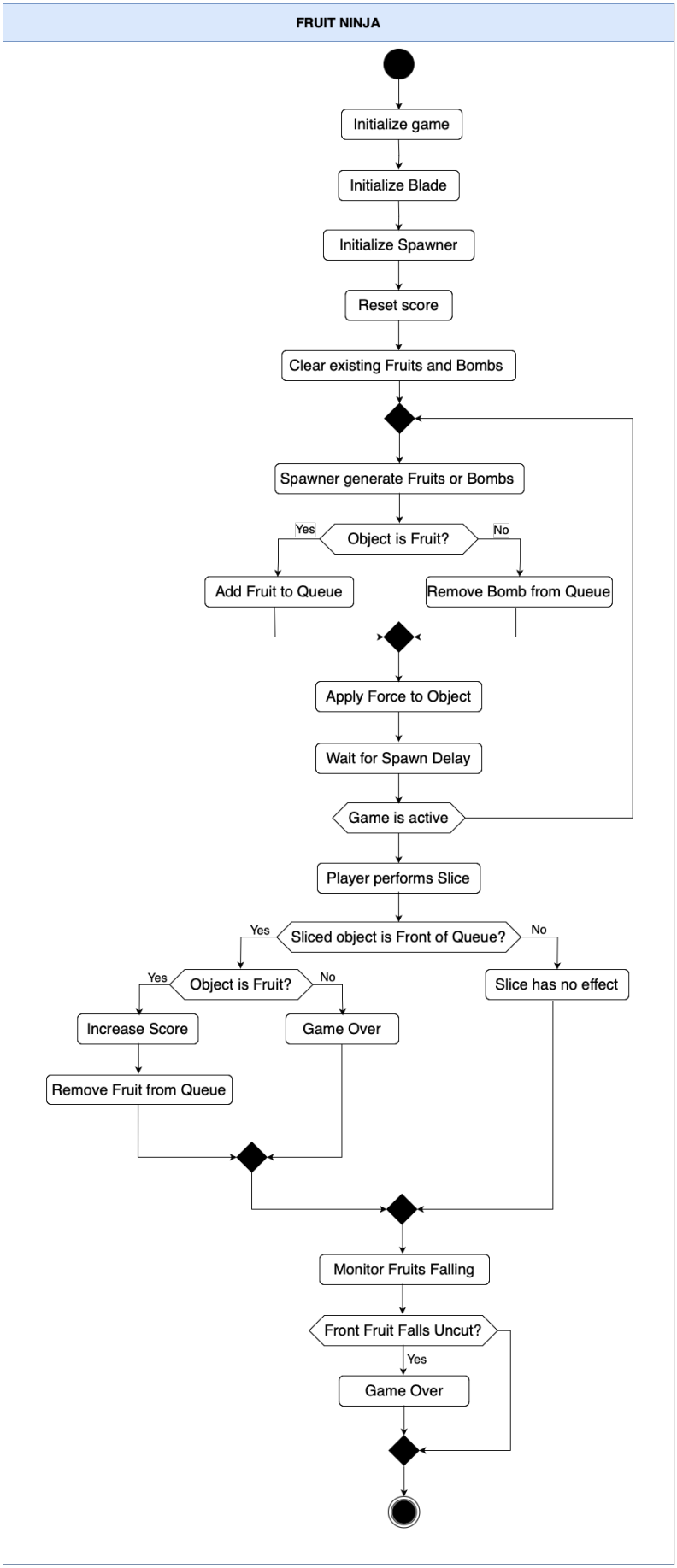

Figure 6: *Activity Diagram of the game*

flow, managing the score, handling game-over conditions, and ensuring that the queue operates smoothly.

Overall, the queue system in Ninja Fruit transforms the game from a simple reflex-based activity into a strategic challenge. Players must not only be quick but also observant and disciplined in following the correct slicing order. This design encourages the development of better observation skills and sustained concentration, providing a more engaging and rewarding gaming experience.

3.4 User Interface Design

The user interface is intentionally kept simple to focus primarily on gameplay. It features a wooden background that creates a cozy and natural atmosphere and has three main elements: blade and fruits and bomb. At the top-left corner of the screen, the player's score is displayed, allowing easy tracking of progress without distraction. To interact with the blade, players can click and drag with the left mouse button on Windows, use a three-finger swipe on macOS, or use touch gestures if their laptop has a touchscreen. This design ensures an intuitive and user-friendly experience across various devices, from desktops to touchscreen laptops. Additionally, there are no other complex UI elements like buttons or toolbars, resulting in a smooth and uninterrupted gaming experience. Keeping the interface simple not only helps players concentrate more on slicing fruits but also makes the game more approachable and accessible to all types of players.

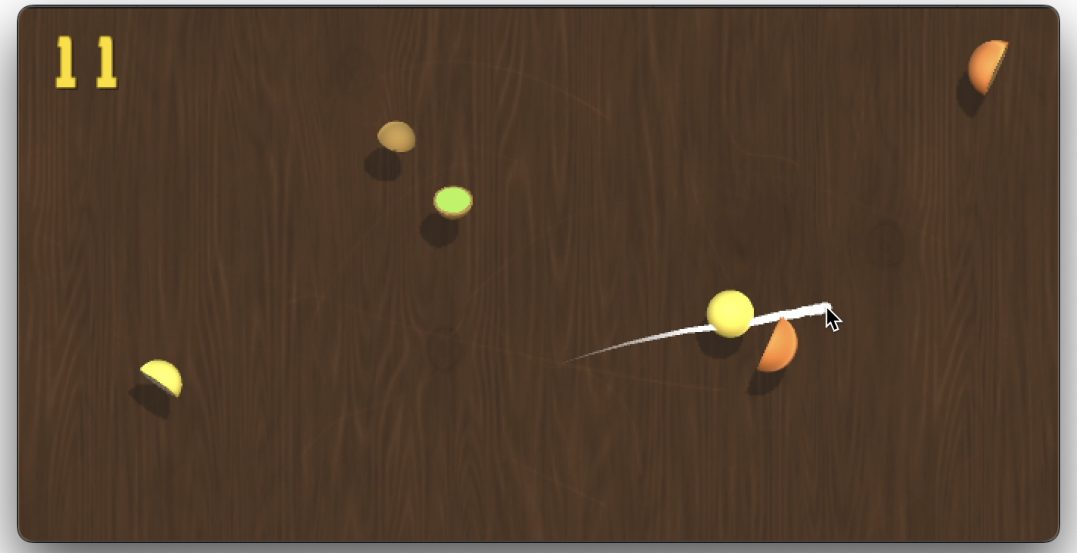

Figure 8: *The interface of the game.*

Furthermore, the colors and images are carefully selected to avoid obstructing the view, ensuring that all important details remain clear and easily identifiable throughout the gameplay. And to make it even more realistic and responsive, we wanted to have some collision effect whenever the blade hit the fruits or bombs, luckily Unity has Bouncy effect so we added it into the effects.

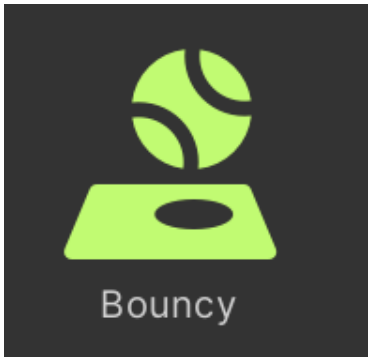

Figure 9: *Main effect of the game.*

3.5 Queue Data Structure in Game Development

In the design and development of the Fruit Ninja game, the queue data structure plays a pivotal role in managing the order of fruits for slicing. This section outlines the theoretical background of queues and their application in this project.

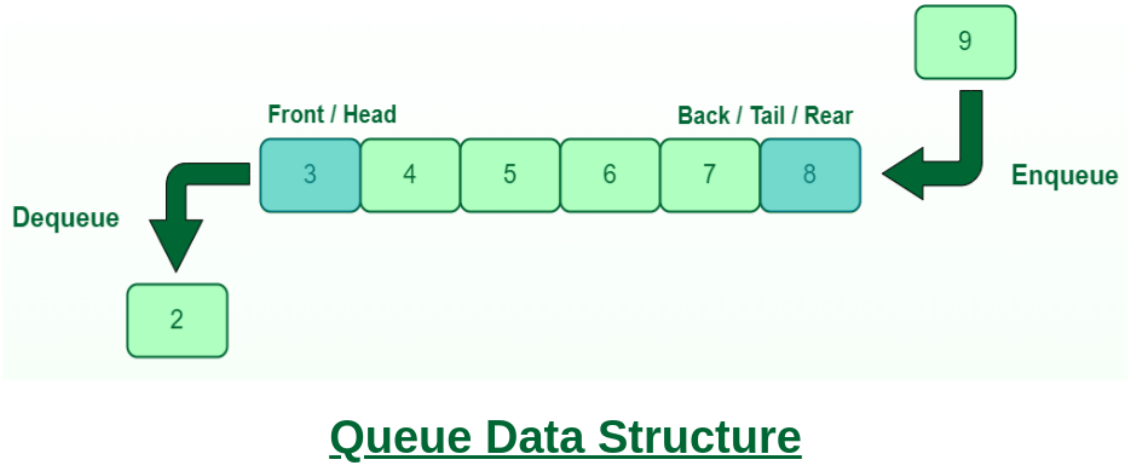

Figure 10: *Queue illustration*

3.5.1 Queue Fundamentals

A queue is a **First-In-First-Out (FIFO)** data structure, where elements are added at the rear and removed from the front. It ensures that the sequence of operations maintains the order of elements as they are inserted.

Key operations of a queue include:

- **Enqueue**: Adding an element to the end of the queue.
- **Dequeue**: Removing the front element of the queue.
- **Peek/Front**: Viewing the front element without removing it.

3.5.2 Application in Fruit Ninja

In this game, the queue is used to manage the fruits appearing on the screen:

- **Enqueuing Fruits**: When new fruits are generated by the Spawner script, they are added to the queue (FruitQueue).

- **Dequeuing for Slicing**: The game restricts slicing to the fruit at the front of the queue. If the player successfully slices the front fruit, it is removed from the queue (Dequeue operation), and the next fruit becomes accessible.

- **Game Over Conditions**: If a fruit at the front is missed (falls off-screen), the game recognizes this as a rule violation, triggering a game-over state. Similarly, attempting to slice a fruit not at the front results in no action, reinforcing the sequential slicing rule.

3.5.3 Benefits of Using a Queue

Order Maintenance: The FIFO property ensures the game logic adheres strictly to the order in which fruits are generated and sliced.
Dynamic Management: The queue dynamically adjusts as fruits are sliced or missed, providing seamless gameplay.
Algorithmic Clarity: The use of queues simplifies the implementation of rules, as all operations align naturally with the game's sequential mechanics.

3.5.4 Enhancing Gameplay

The integration of a queue transforms Fruit Ninja from a simple reflex-based game to one that challenges players to observe and strategize. Players must carefully track the sequence of fruits and react in real-time, adding depth to the gaming experience.

4 Development

The development of a Fruit Ninja-style game using Unity 2D, a versatile and powerful game engine, provides a rich and interactive slicing experience. Unity's comprehensive toolset streamlines the creation of core game mechanics, visual assets, and responsive user interactions. The development process begins with the design of sprite assets, including vibrant and detailed fruit models, bombs, and backgrounds. These assets are imported into Unity, where developers use the scene editor to arrange gameplay elements and the particle system to enhance visual effects like juice splashes and explosions. Game mechanics are implemented through C# scripting, managing fruit generation, slicing detection, and scoring logic. Scripts control the trajectory of fruits, the detection of player gestures, and game events like combo rewards or penalties for slicing bombs. Unity's physics engine plays a key role in creating realistic fruit arcs and interactions, while animation tools bring the game to life with dynamic slicing effects and smooth transitions. Explosive feedback for bombs and special effects for power-ups are integrated for added immersion. The UI system in Unity ensures a seamless player experience, with responsive menus, score counters, and tooltips. These elements are designed to be visually engaging and intuitive, catering to both casual and competitive players. Overall, Unity 2D's features enable the efficient development of

an engaging and polished Fruit Ninja-style game, delivering a visually stunning and responsive gameplay experience.

4.1 Initial setup

The project will start by creating a simple 2D project in Unity. Here, the environment of Unity is initialized, with the necessary parameters set correctly to develop in a 2D game, and the workspace is readied for the import and management of assets.

4.2 Create some necessary folders and files

4.2.1 Prefabs

The Prefabs folder contains reusable game objects that form the foundation of the Fruit Ninja game. These prefabs include fruits like Apple, Kiwi, Lemon, Orange, and Watermelon, as well as a Bomb. Each prefab is pre-configured with properties such as colliders and Rigidbody components for physics-based interactions, making them ready for instantiation during gameplay.

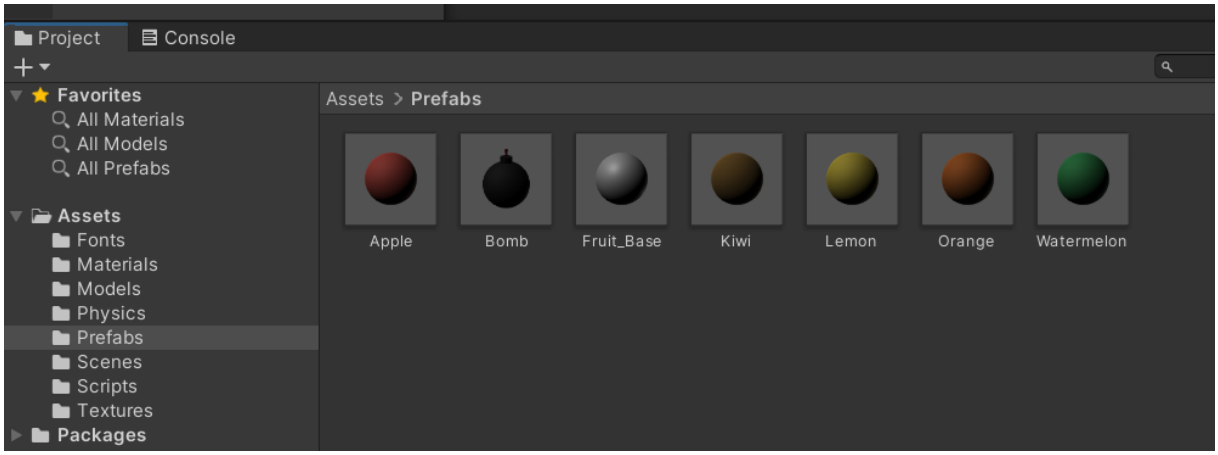

Figure 11: *"Prefabs" folder*

4.2.2 Materials

Prefabs also reference Materials stored in the "Materials" folder, which define the visual appearance of the objects. Materials control the surface properties, such as color, texture, and smoothness, ensuring that objects look realistic and visually distinct. This separation allows developers to independently update the look of objects through material adjustments while keeping the prefab logic intact, providing flexibility and efficiency in development.

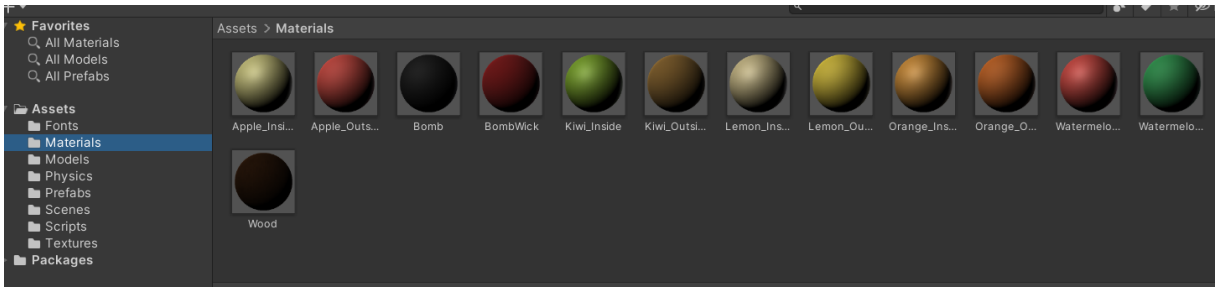

Figure 12: *"Materials" folder*

4.2.3 Models

The Models folder contains the 3D assets used to define the geometry of objects in the game, such as fruits. These models form the foundation of the visual representation of fruits and their sliced components. The Fruit_Whole model represents the unsliced fruit as it appears when spawned in the game, while the Fruit_Sliced model is used to depict the fruit after it has been sliced by the player.

Additionally, models like Top, Bottom, Inside, and Outside represent individual parts of the sliced fruit, such as the inner flesh and outer skin, which are combined to create a realistic slicing effect. The Sphere model, a basic 3D object, may serve as a template or placeholder for creating custom fruit models or be used for physics-based calculations. These models are paired with materials from the "Materials" folder to achieve realistic textures and colors, and they are incorporated into prefabs with physics components to enable interactions and dynamic behavior during gameplay. This structure ensures both visual accuracy and smooth game mechanics.

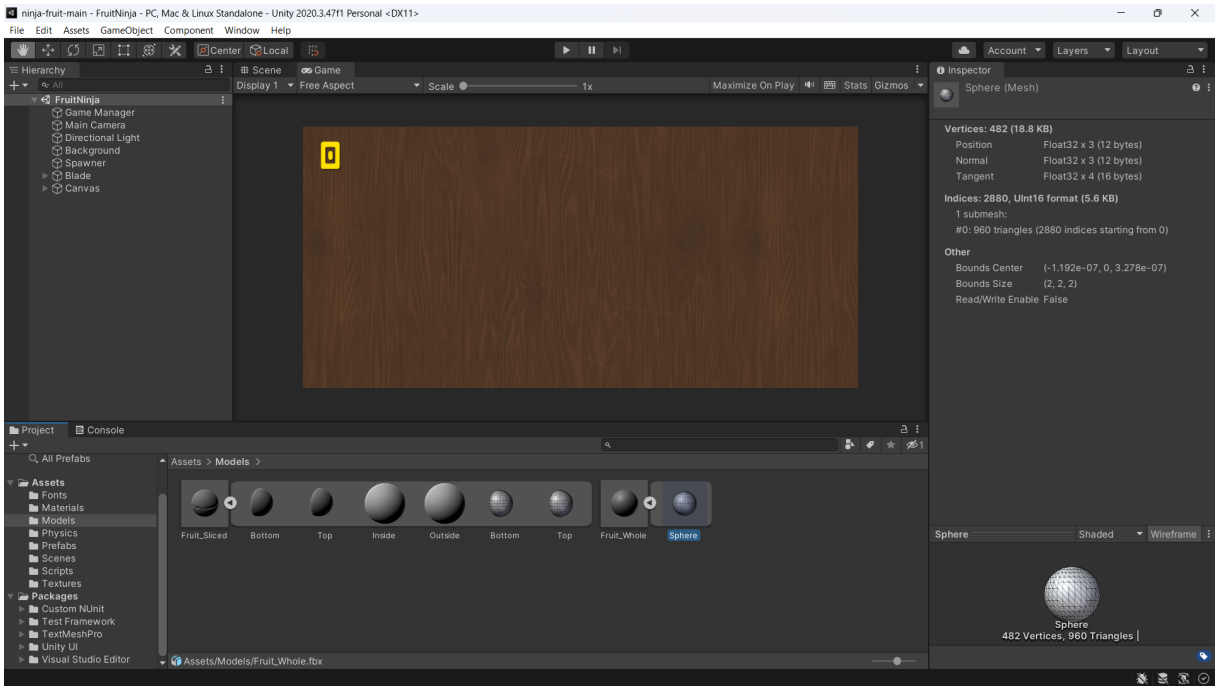

Figure 13: *"Models" folder*

4.2.4 Textures

This folder contains a photo of the main UI of the game:

Figure 14: *"Wood" photo (The main UI of Fruit Ninja)*

When we open the game, it looks like this:

Figure 15: *"Wood" photo (The main UI of Fruit Ninja)*

4.2.5 Scripts

This folders includes all necessary C# files for Fruit Ninja:

- **Fruit**:
  The Fruit script is designed to manage the behavior of individual fruits in the Fruit Ninja game. It handles their appearance, slicing mechanics, and interaction with the game environment. This script combines visual effects, collision detection, and game logic to ensure fruits behave dynamically and contribute to the gameplay experience.

  Each fruit has two states: whole and sliced. These are represented by two GameObjects, whole and sliced, that are toggled based on whether the fruit is intact or has been cut. The Slice method is triggered when the fruit collides with the blade, and it handles the slicing process. This includes switching to the sliced model, playing a juice particle effect, and applying physics to the slices. The slicing direction and force are passed from the blade, giving the sliced pieces realistic motion.

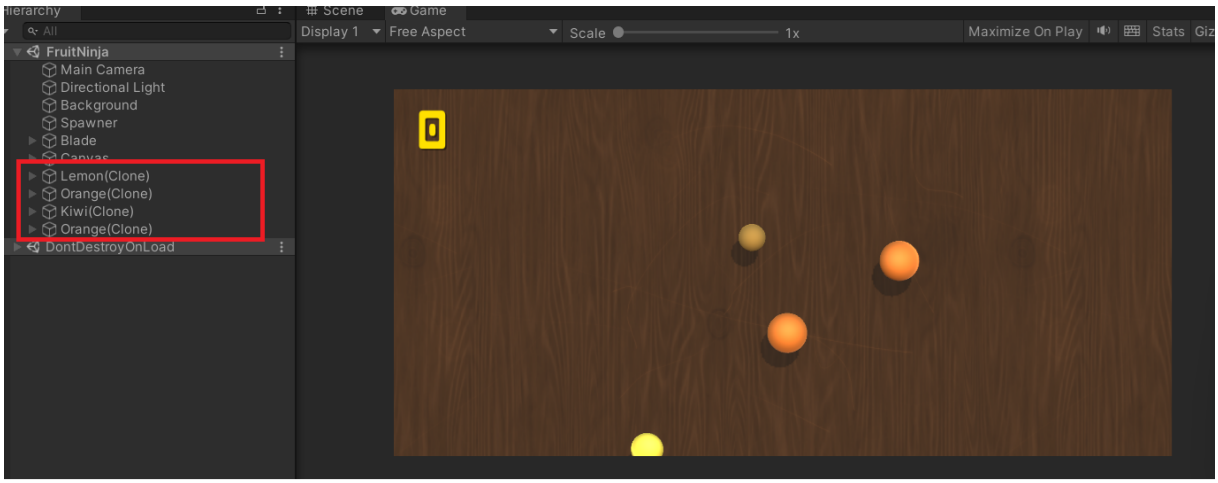

Figure 16: *Some fruits show up on the screen*

The script integrates closely with the game manager to ensure proper scoring and game progression. When a fruit is sliced, it checks if it is the first fruit in the FruitQueue. If so, it increases the player's score and removes the fruit from the queue. This ensures that the gameplay follows a logical sequence and prevents unintended interactions. A flag (isSliced) is also used to ensure a fruit cannot be sliced multiple times, maintaining consistency in scoring and effects.

Additionally, the script includes functionality to detect when a fruit falls off the screen. It tracks the fruit's vertical position and checks if it moves below the viewport. If this happens and the fruit has not been sliced, the game manager is notified to trigger a game-over state. This feature ensures that missed fruits contribute to the game's challenge.

- **Blade**:
  The Blade script is a key component of the slicing functionality in the Fruit Ninja game, enabling players to cut fruits using mouse movements. The script is built around handling user input, tracking blade movement, and detecting collisions with game objects. The slicing action is initiated when the player presses the left mouse button, which starts the blade movement. The script positions the blade at the mouse cursor in the game world using the main camera to convert screen-space mouse coordinates into world-space positions. This ensures the blade's movement aligns seamlessly with the player's input.

  The slicing effect is achieved through the use of a collider and a trail renderer. The collider is enabled during slicing to detect intersections with objects, such as fruits, and is dynamically activated or deactivated depending on the blade's velocity. This ensures that the blade only slices when moving fast enough, as determined by the minSliceVelocity parameter. Meanwhile, the trail renderer adds a glowing visual effect to the blade's movement, enhancing the gameplay experience by visually representing the slicing action.

Figure 17: *The blade cutting fruits*

When the slicing begins, the blade tracks the direction and velocity of its movement. This is achieved by calculating the change in position over time, allowing the script to determine whether

the velocity exceeds the slicing threshold. If it does, the collider remains active, allowing the blade to interact with fruits. The slicing stops when the player releases the mouse button, at which point the collider and trail renderer are disabled, ensuring efficient resource usage and precise control over the slicing mechanism.

- **Bomb**:
  The Bomb script is a straightforward component in the Fruit Ninja game that handles the behavior of bombs when they interact with the player's blade. Its primary purpose is to detect collisions and trigger the game-over sequence.

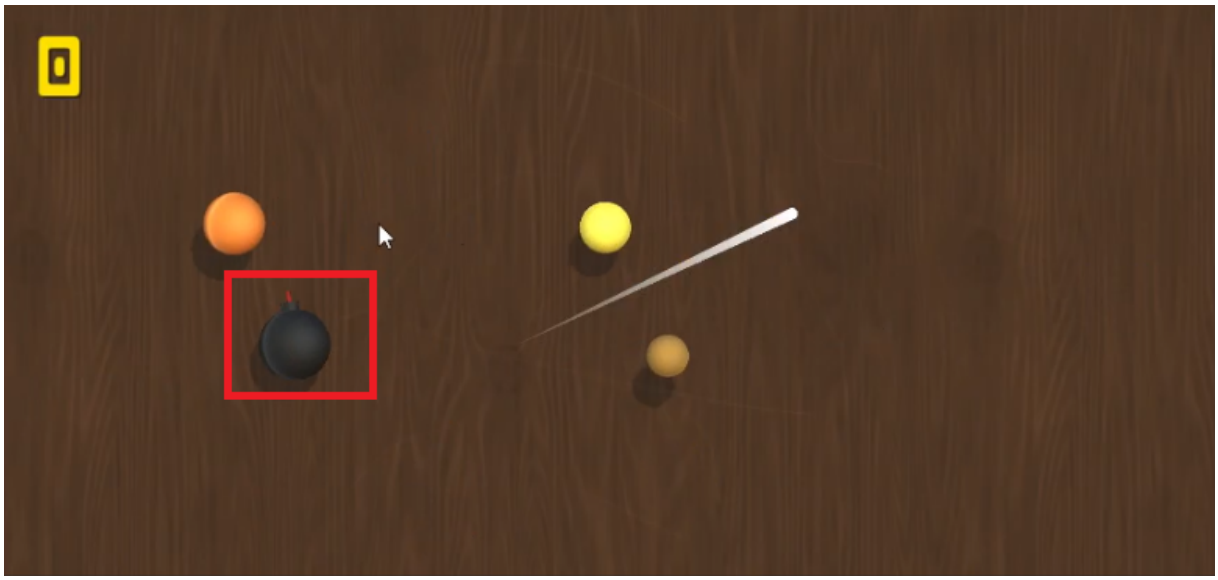

Figure 18: *The bomb that players have to avoid cutting*

When the OnTriggerEnter method is called, it checks if the object it collided with is tagged as "Player," which represents the blade. If the collision is confirmed, the script disables the bomb's collider to prevent further interactions and immediately calls the Explode method from the GameManager instance. This action is responsible for triggering the game's failure state, such as displaying a game-over screen or stopping the game.

The simplicity of the script ensures that bombs are purely detrimental obstacles in the game. Players must avoid slicing bombs, as interacting with them leads to an immediate loss.

- **Spawner**: The Spawner script is central to creating dynamic gameplay in Fruit Ninja, as it handles the random spawning of fruits and bombs during the game. It ensures objects appear unpredictably with varying positions, angles, and velocities, challenging the player to react quickly. The script also incorporates a queue to maintain the sequence of spawned fruits, which is particularly useful for implementing mechanics where players must slice fruits in a specific order.

  The script uses a collider (spawnArea) to define the boundaries within which objects will spawn. It contains arrays for fruit prefabs (fruitPrefabs) and a bomb prefab (bombPrefab), which serve as templates for creating instances of fruits and bombs. Adjustable properties, such as minSpawnDelay, maxSpawnDelay, minForce, maxForce, and minAngle to maxAngle, allow developers to control the timing, velocity, and trajectory of the spawned objects. Additionally, the bombChance

variable introduces a probability that a bomb will replace a fruit, adding an element of risk and complexity to the game.

A notable feature of the script is the FruitQueue, a queue that keeps track of the order in which fruits are spawned. Every new fruit is added to this queue, enabling the game to implement rules that depend on slicing fruits in the correct sequence. This system adds a layer of strategy to the game, as players may need to prioritize certain fruits over others to maximize their score.

The spawning process is handled by a coroutine (Spawn) that continuously runs while the script is enabled. During each iteration, the coroutine selects a random prefab, determines its spawn position and rotation within the defined area, and instantiates it. An upward force is applied to the object's rigid body to simulate the action of being thrown into the air. Objects are destroyed after a maximum lifetime to avoid clutter and manage performance. Randomized spawn delays ensure the timing of appearances remains unpredictable, keeping players engaged.

Finally, the OnEnable() and OnDisable() methods control the lifecycle of the spawning process. The coroutine starts when the script is enabled and stops when it is disabled, ensuring objects are only spawned while the game is active. This design ensures efficient resource management and seamless integration into the game.

- **GameManager** The GameManager script serves as the central controller for the Fruit Ninja game, managing the game's overall state, scoring system, and interactions between other scripts such as Blade and Spawner. It utilizes the singleton pattern to ensure there is only one instance of GameManager throughout the game. The Awake() method initializes this singleton and ensures the GameManager persists across scenes by using DontDestroyOnLoad.

  The NewGame() method resets the game state at the start or after a game over. It clears the scene of any existing fruits or bombs, re-enables the Blade and Spawner scripts, and resets the player's score. The score is displayed using a Text UI component, and the FruitQueue—used to manage the sequence of fruits for gameplay mechanics—is cleared to prepare for new spawns.

  The ClearScene() method ensures a clean state by destroying all active fruit and bomb objects in the scene. This method is particularly useful when restarting the game to prevent overlapping objects from previous sessions, maintaining consistency and performance.

  The scoring system is handled by the IncreaseScore() method, which updates the player's score and displays it. The method also compares the current score with the high score stored in PlayerPrefs. If the current score surpasses the high score, it updates the stored value. This provides a persistent way to track the player's best performance across sessions.

  The Explode() method triggers the game over sequence, disabling the Blade and Spawner scripts to stop gameplay. It initiates a coroutine called ExplodeSequence() that handles a visually dynamic transition when the game ends. This sequence gradually fades the screen to white while slowing down the game using Time.timeScale. After a brief pause, the game restarts by calling

NewGame(), and the screen fades back to normal.

The ExplodeSequence() coroutine provides a polished end-game effect by fading the screen in and out smoothly. It uses Mathf.Clamp01 and Color.Lerp to interpolate the fade color, creating a visually appealing transition. The time scale manipulation adds a slow-motion effect, enhancing the dramatic impact of the game-over moment.

Additionally, the GameManager acts as a central point for accessing the FruitQueue, which is managed by the Spawner script. By exposing this queue as a property, it allows other scripts to interact with it, such as ensuring fruits are sliced in the correct order. This integration supports the game's mechanics and enforces rules that keep the gameplay engaging.

4.3 Testing

During testing of the Fruit Ninja game, the primary goal was to ensure that all gameplay mechanics were functioning correctly, with particular attention to the slicing of fruits, the behavior of bombs, and the overall game flow.

First, we simulated fruit spawns using the Spawner script, verifying that fruits were appearing at random intervals and positions within the designated spawn area. We also tested the bomb mechanics by ensuring that bombs were randomly spawned and, when sliced, triggered the game over sequence. The blade's slicing functionality was thoroughly examined by mimicking user input, ensuring that the Blade script properly responded to mouse movements, and that fruits were sliced when they intersected with the blade's path. The scoring system was tested by slicing fruits and confirming that the score updated correctly. Additionally, the game's transition between game over and restart was tested, ensuring that the fade-in and fade-out effects worked smoothly, and that the game state was properly reset after an explosion. The handling of fruit falling off-screen was also verified to trigger a game over.

All aspects of the game were tested across multiple sessions to ensure stability, responsiveness, and a seamless user experience.

5 Results

We have created successfully flawless **Fruit Ninja** game following **Queue - First In First Out** from Data Algorithms and Structure that we have studied before, that means players have to cut the fruits showing up on the screen respectively. If players don't follow the rules, the other fruits can't be cut leading to losing the game.

Here are some screenshots of the full gameplay:

1. After we start the game, we see that there's a first fruit - Watermelon showing up, we have to cut it first, then the other (Apple) respectively.

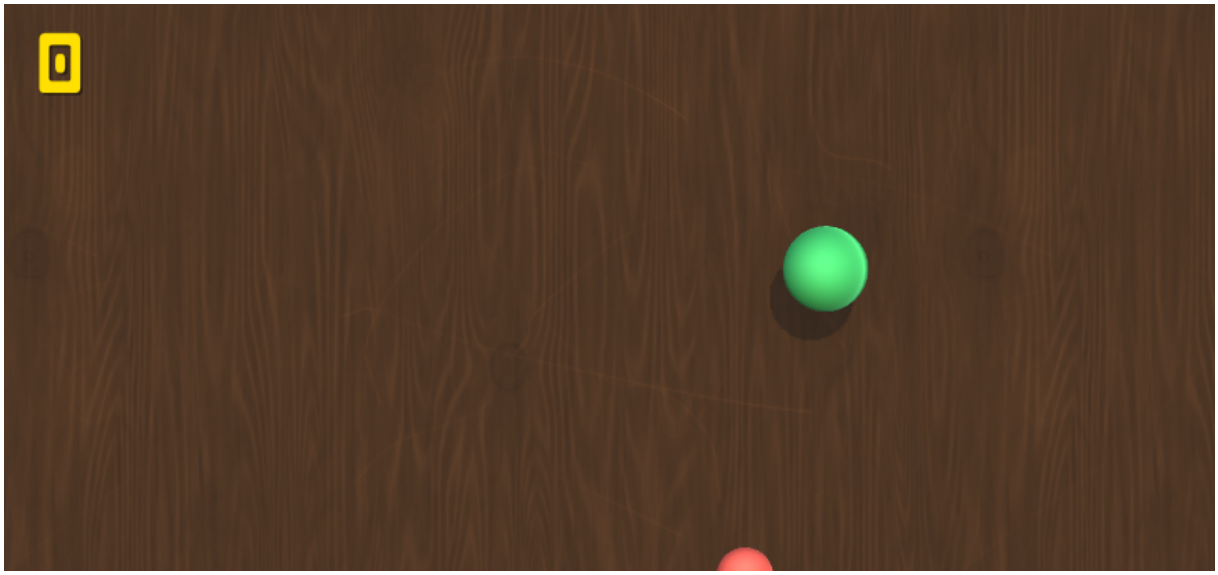

Figure 19: *The first fruit - Watermelon, and the second one - Apple*

2. After cutting the Watermelon then Apple, we've got 2 points from them. We keep cutting to get high scores as much as possible.

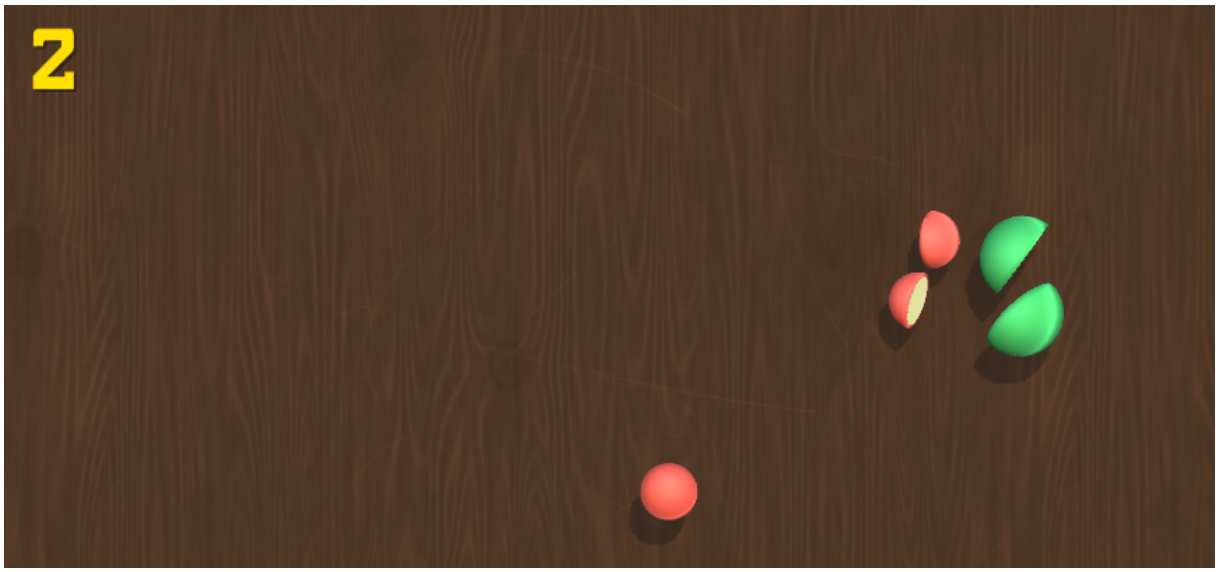

Figure 20: *Got 2 points after cutting Watermelon and Apple in order*

3. Let's talk about **game losing**. We have 2 ways to lose the game: **Slice the bomb** or **Miss one fruit**. Regarding the bomb, we can see there's a bomb on the screen, we have to avoid cutting it.

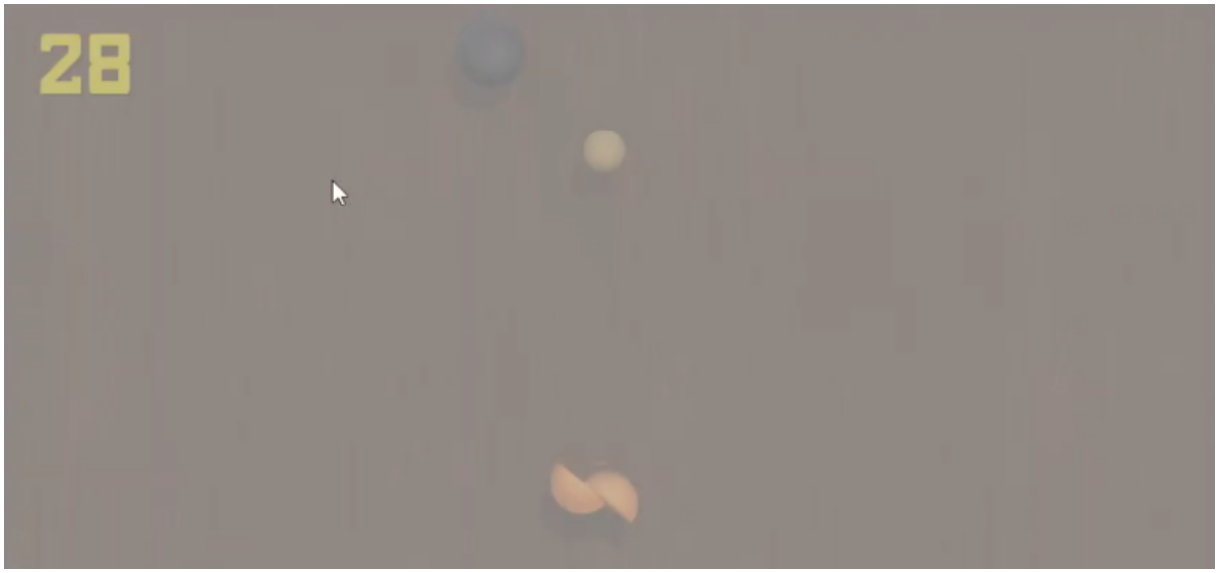

Figure 21: *Game losing* - ***Slice the bomb***

If we slice the bomb like this, the screen will change the light to white, then we can replay the game.

About the last one (Miss one fruit), if we don't follow the rules - Cut in order, it will lead to game losing like this:

Figure 22: *The order of fruits*

Figure 23: *Game losing* - ***Miss one fruit***

In conclusion, the development of our Fruit Ninja-style game has been an exciting journey, blending creativity with technical challenges to emulate the thrill of fast-paced, skill-based gameplay. From initial design to the final implementation, the process was both demanding and rewarding.

During the design phase, we focused on creating a clear and intuitive class diagram to capture the core mechanics of the game in a way that would support future development. We also designed vibrant, eye-catching sprites and animations to deliver a visually engaging and dynamic slicing experience that players would find enjoyable and immersive.

In the development phase, while we couldn't fully realize every aspect of our original concept, we believe we came very close. Learning to combine our new knowledge of Unity with C# scripting to manage fruit generation, slicing detection, and interactions was challenging but deeply satisfying. Staying true to the framework and planning established during the analysis phase played a crucial role in keeping us on track.

This project has offered us valuable insight into the gaming development process, particularly when using Unity as a game engine. Despite the project's limited scope, we have learned essential lessons and gained practical experience that we can apply to future endeavors. This game serves as a strong foundation for continued growth and more ambitious projects in the future.

6 Git Manuals

Here is the instructions to use our game:

1. Install **Unity**.
2. Install **Unity Editor v2020.3.47f1** (Recommended).
3. Clone our respiratory from **Github** by clicking this link.

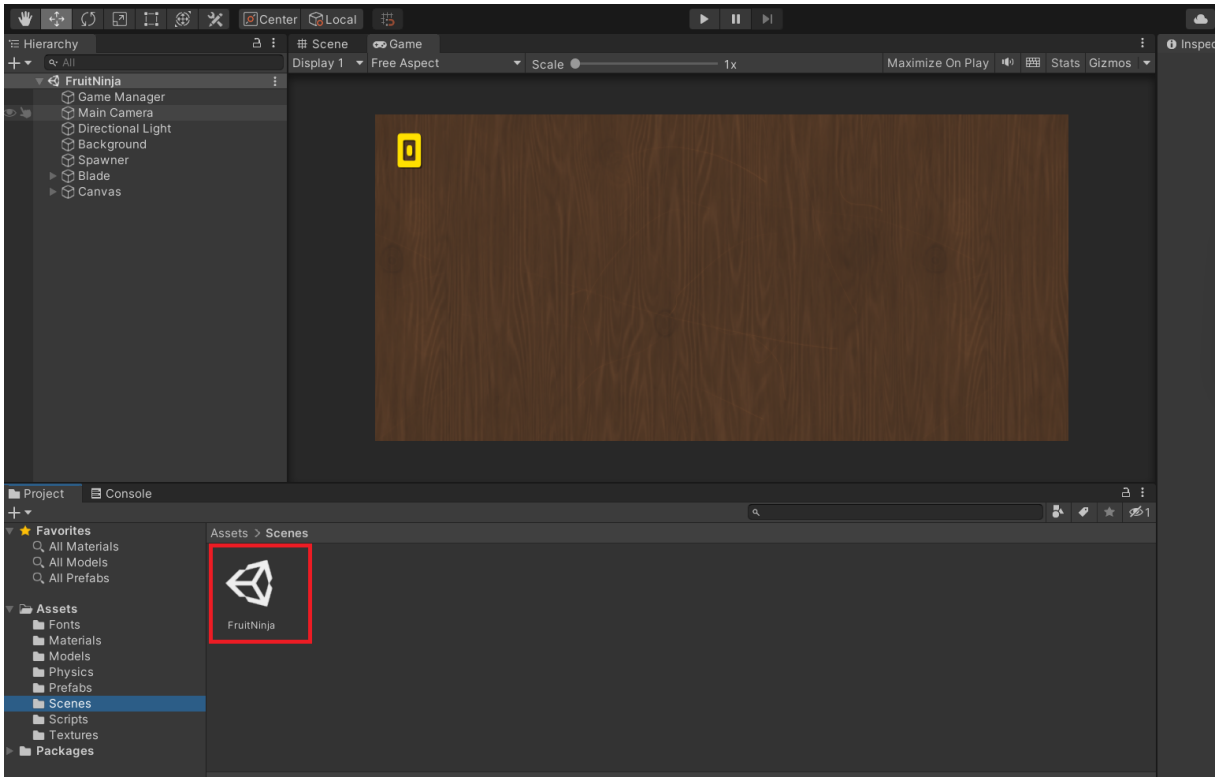

Figure 24: *Fruit Ninja project*

4. After importing the project, the game will be opened like this: Then we just click the file **FruitNinja** from **Scenes** folder. After that, the game is ready for us to play and enjoy.