Gisela Borràs i Héctor Núñez
VJ Q2 2024 - Upc FIB
Tutored by

FIB
UPC

02/06/2024

Table of content

1. About the original game

Temple Run, originally released in 2011, was developed and published by Imangi Studios, a small independent game development studio founded by Keith Shepherd and Natalia Luckyanova. Since its initial launch, the game has seen numerous versions and updates, including sequels like Temple Run 2, Temple Run: Brave, and Temple Run: Oz, each adding new features and enhancements to the gameplay experience.

Initially available on iOS devices, Temple Run has expanded its reach to Android, Windows Phone, and other platforms, ensuring a broad audience across various mobile ecosystems. The game's primary appeal lies in its simple yet addictive gameplay, where players control an explorer running through an ancient temple, avoiding obstacles and collecting coins. The endless runner format, combined with intuitive swipe controls, has made it a favorite among casual gamers.

Targeted primarily at mobile gamers of various age groups, Temple Run has achieved remarkable popularity, being downloaded more than 500 million times. With a significant presence on Game Center, boasting around 50 million gamers, Temple Run remains one of the most played applications and consistently ranks in the top 50. Its widespread appeal and success were further highlighted when it was recognized in the Nickelodeon Kids' Choice Awards of 2013 in the Favorite Application Category.

The game's influence extends beyond just downloads and awards. Temple Run has inspired a genre of endless runner games and has been credited with helping to popularize the freemium model in mobile gaming, where the game is free to download but offers in-app purchases. This model has since become a standard in the mobile gaming industry, demonstrating the significant impact Temple Run has had on the market.

Overall, Temple Run's combination of engaging gameplay, broad accessibility, and innovative business model has cemented its place as a landmark title in the world of mobile gaming, continuing to attract players and inspire new game developers around the globe.

2. Project description

2.1. Global description

The objective of Medieval Run is to stay alive for as long as possible while navigating through a dangerous passageway over water. Players must avoid various obstacles such as gaps, bridges, cacti, and rocks by swiping, jumping, or turning. Since it is an endless runner game, there is no endpoint to the level, allowing players to run indefinitely as long as they avoid the hazards. This infinite format means that there is only a single, continuously evolving level, which increases in difficulty the longer the player survives. As the player progresses, they can collect coins and power-ups that temporarily grant special abilities.

At any time, players have the option to activate an invincible mode, known as God Mode, which will trigger a golden shield [Fig. 9], indicating the player's invincibility. This feature is activated and deactivated by pressing the G key.

2.2. HUD

In the game, the Heads-Up Display (HUD) is designed to provide the player with essential information at a glance, ensuring they can easily keep track of their progress and resources. Positioned at the top right corner of the screen, the HUD features two main elements: a distance counter and a coin counter.

The distance counter, located on the left, displays an icon of a trophy. This element is crucial for tracking the distance the player has traveled during the game. The distance traveled often symbolizes the player's progress, motivating them to keep moving forward and reach new goals.

Next to the distance counter is the coin counter. This element features an icon of a coin with a star in the center. The coin counter is vital for monitoring the amount of currency the player has accumulated, which can be used for upgrades, purchases, or unlocking new content within the game.

2.3. Technical description about main player, enemies and obstacles

2.3.1. Main Player

The `MainPlayer` script in Unity is a comprehensive control system for the main character, Tom. This script manages a wide range of behaviors including movement, jumping, animations, health management, and interactions with the environment.

At its core, the script handles Tom's forward movement by controlling his speed and acceleration. Tom's movement speed gradually increases to a specified maximum, creating a smooth acceleration effect. This is complemented by the ability to strafe left and right, allowing Tom to navigate horizontally. The strafing movement also employs a gradual acceleration, ensuring smooth transitions. Jumping is another key aspect, with the script applying a defined force to make Tom jump. Ground detection is handled through raycasting, which ensures that jumping and other grounded actions only occur when Tom is on the ground.

Health management is a crucial part of the script, with Tom starting with a set number of lives. The script keeps track of these lives and triggers a death sequence when they run out. During gameplay, Tom can collect power-ups like shields, which can temporarily increase his maximum lives, and magnets, which attract coins to him. These power-ups are visually represented by particle effects, enhancing the gameplay experience.

Tom's interactions with the environment are also managed meticulously. The script includes functions for handling collisions, attacks, and other interactions. For instance, when Tom initiates an attack, the script randomly selects one of two attack animations to play, adding variety to his actions. The script also manages animations for other actions like jumping and rolling, ensuring that Tom's movements are visually represented accurately.

In addition to controlling Tom's actions and interactions, the script also manages the scoring system by tracking the distance traveled and the number of coins collected. This data is used to adjust the game speed dynamically, increasing the challenge as the player progresses. The script also includes functions to restart the scene or return to the main menu, providing essential game management features.

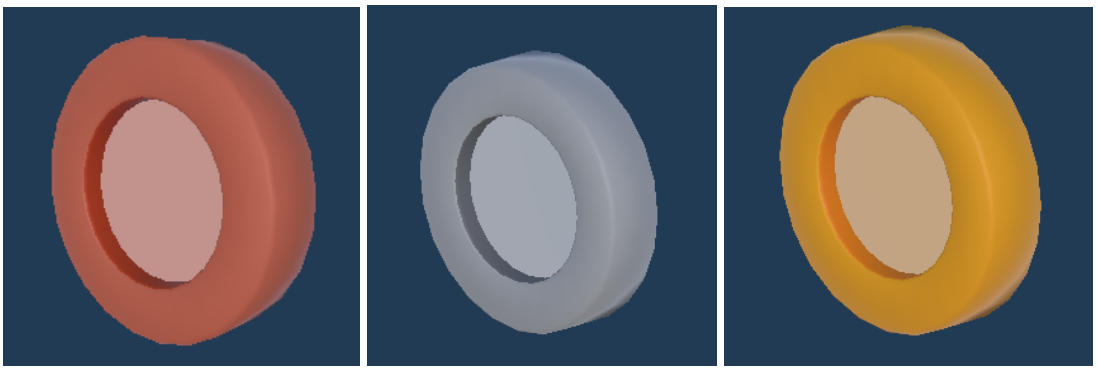
Overall, the `MainPlayer` script is a robust and versatile control system that ensures Tom's actions and interactions are smooth, responsive, and engaging. By integrating movement controls, health management, environmental interactions, and power-ups, the script provides a comprehensive framework for managing the main character in a Unity game.

2.3.2. Coins

In the game, there are three types of coins that players can collect, as shown in the image. Each type of coin has a distinct color, providing visual variety and different point values. Additionally, there are special star coins for each type, which offer more points when collected. The regular coins come in three variations:

1. Bronze Coin: A standard coin that adds a basic amount of points to the player's score.
2. Silver Coin: The most valuable regular coin, providing the highest number of points among the three.
3. Gold Coin: A slightly more valuable coin that adds more points than the red coin.

For each of these coin types, there is a corresponding star coin. Collecting a star coin not only adds the regular point value but also gives an extra bonus, significantly boosting the player's score.



2.3.3. Power Ups

2.3.3.1. Magnet Power-Up

The Magnet power-up attracts nearby coins to the player, making it easier to collect them without having to move directly over them. The CoinAttractor script controls this behavior. When the player collects the Magnet, the script activates an attractor force that pulls coins within a certain range towards the player. The strength of this attraction can be adjusted with the attractorStrength parameter. The Magnet power-up is divided into three levels, each with increasing effectiveness, controlled by the Magnet script. Depending on the player's current magnetic level, different versions of the Magnet (MagnetLvl1, MagnetLvl2, MagnetLvl3) are activated. When the player collides with the Magnet power-up, it triggers an audio effect, adds the Magnet effect to the player, and then destroys the power-up object, generating a particle effect for visual feedback.

2.3.3.2. Shield Power-Up

The Shield power-up provides the player with temporary protection against obstacles and enemies. Similar to the Magnet, the Shield power-up is managed by a script that activates different levels of shield strength (ShieldLvl1, ShieldLvl2, ShieldLvl3) based on the player's current shield level. The Shield script handles the activation of the shield when the player collects the power-up. Upon collision with the Shield power-up, the player gains a protective shield, an audio effect is played, and the power-up object is destroyed, leaving behind a particle effect. This shield allows the player to survive hits that would normally result in damage or game over, significantly enhancing their chances of progressing further in the game.

2.3.3.3. Implementation Details

Both the Magnet and Shield scripts include logic to initialize the appropriate level of the power-up based on saved game data, ensuring that players receive the correct level of enhancement. The OnTriggerEnter method in both scripts handles the interaction with the player, triggering the effects and playing corresponding audio cues. The Kill method is responsible for creating the particle effects and removing the power-up object from the game world, providing a satisfying visual and auditory experience.

The PowerUpSpawner script handles the generation of power-ups, which provide the player with temporary abilities or bonuses. Similar to obstacles, power-ups are defined using the PowerUpData structure, which includes the prefab and width of each power-up. The script randomly selects a power-up from the list, determines a spawn position within the specified range, and instantiates it. Power-ups are placed strategically to encourage exploration and add an element of reward to the gameplay.

2.3.4. Destructible Obstacles

In the game, there are various types of destructible objects that the player encounters along the path. When the player character collides with one of these destructible objects, the pursuing zombies get closer to the player. This proximity increases the danger, as a second collision with any of these objects will result in the player's immediate death, leading to a game over.

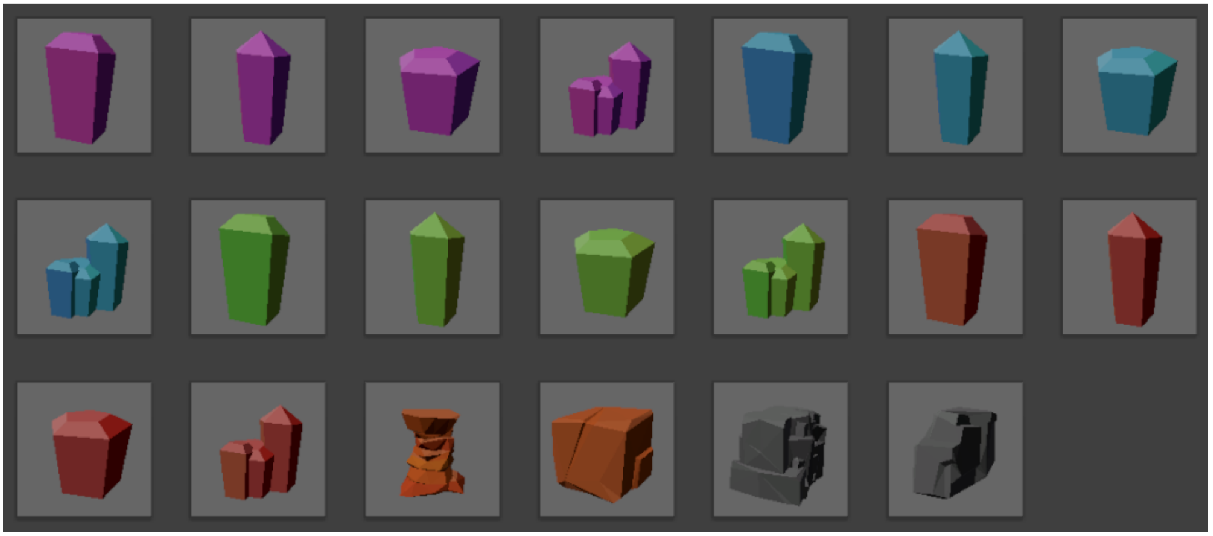
To avoid these obstacles, the player has two main options. The first option is to skillfully dodge the obstacles, using precise movements to avoid collisions entirely. This requires good reflexes and control, as the player must navigate the environment carefully to maintain their distance from the pursuing zombies.

The second option provides a more direct approach: the player can press the spacebar to use the stick their character carries. This action breaks the destructible object, clearing the path and allowing the player to continue without taking damage. This mechanic introduces a strategic element, as the player must decide when to use the stick to break objects and when to rely on their dodging skills.

2.3.5. Non-destructible obstacles

In the game, non-destructible objects play a crucial role in shaping the player's navigation and strategy. These objects cannot be broken or moved by the player, serving as static obstacles that must be skillfully avoided to ensure survival and progression. The variety of non-destructible objects not only adds visual diversity to the game but also enhances the challenge by requiring the player to constantly adapt and refine their movements.

One type of non-destructible object is the crystals, which come in several colors and shapes. Purple crystals are tall and slender, often found in pairs or clusters, creating narrow passages that the player must carefully navigate through. Blue crystals, similar to the purple ones but with a striking blue color, add variety to the landscape and require the player to stay alert. Green crystals come in different shapes and sizes, presenting varying levels of difficulty as the player maneuvers around them. Red crystals are larger and more prominent, providing a significant visual contrast and blocking larger portions of the path.



Rocks are another common non-destructible obstacle. Standard rocks are irregularly shaped and scattered along the path, necessitating quick adjustments in the player's route to avoid collisions. Stacked rocks, arranged in a tower-like formation, can create narrow spaces or blind spots, requiring precise navigation. Large boulders are larger and more obstructive, demanding careful planning and quick reflexes to dodge.

Adding a unique challenge is the elevated wooden bridge. This structure introduces a vertical element to the obstacle course. The elevated bridge spans across the path, and the player must slide underneath it to avoid collision. This requires executing a specific slide move, adding a layer of timing and precision to the gameplay. Failing to slide in time results in a collision, increasing the stakes and testing the player's reflexes.

The presence of these non-destructible objects enhances the gameplay by requiring players to continuously navigate around them. Unlike destructible objects that can be cleared from the path, these static obstacles remain in place, forcing players to develop quick reflexes and precise control. Players must use precise movements to dodge these static obstacles, ensuring they do not collide with them. Collisions with non-destructible objects can lead to significant setbacks or even game-over scenarios if they cause the player to lose balance or crash. For obstacles like the elevated wooden bridge, players need to perform a sliding action to safely pass underneath, adding a vertical navigation challenge that tests the player's timing and reflexes.

2.3.6. Decoration

In the game, decoration objects significantly enhance the visual appeal and immersion of the environment. These objects include a variety of natural elements, such as rocks, boulders, bushes, shrubs, and grass tufts, which add texture and detail to the landscape. Additionally, man-made elements like wooden planks, beams, and railway tracks contribute to the narrative and setting by suggesting past human activity or industrial infrastructure. The `DecorationSpawner` script is responsible for placing these objects dynamically within the game world. It uses predefined probabilities and spawn ranges to ensure a varied and natural distribution. The script employs raycasting to confirm that decorations are placed on valid ground surfaces, preventing floating or misplaced objects. Each decoration is assigned a random rotation to enhance realism and avoid uniformity. This combination of natural and man-made elements creates a visually rich and engaging environment that encourages players to explore and enjoy the game world.

The DecorationSpawner script is tasked with adding aesthetic elements to the game environment. It spawns decorations like trees, rocks, and other environmental objects at random positions within a specified range. The script uses raycasting to ensure that decorations are placed on valid ground surfaces. Each decoration has a chance to spawn based on a predefined probability, ensuring that the environment is populated in a balanced and visually appealing manner. The random rotation applied to each decoration further enhances the natural look and feel of the game world.

2.3.7. Enemies

The EnemiesFollow script in Unity manages the behavior of three enemy characters that follow a main player character. The script references three enemy GameObjects and their respective animators, as well as a MainPlayer script attached to the player GameObject. It controls the positioning and animations of the enemies based on the player's status and actions. The enemies adjust their position relative to the player based on the player's health (lives), ensuring they maintain a specific offset distance. If the player's lives decrease, the enemies move closer, while if the player's lives are more than one, the enemies maintain a farther distance. The script also smoothly transitions the enemies' positions using linear interpolation (Lerp) for a smooth following effect.

In addition to following the player, the script handles enemy animations in response to collisions and the player's state. If the player is dead, the enemies stop running and switch to a "Cast Spell" animation. If the enemies are not colliding and not jumping, they trigger a jump animation. The script also includes a method, individualOffsets, to adjust each enemy's position individually when collisions are detected, moving them slightly forward or backward to avoid overlapping. This ensures that the enemies maintain their relative positions and animations appropriately, providing a dynamic and engaging gameplay experience.

2.3.8. Camera

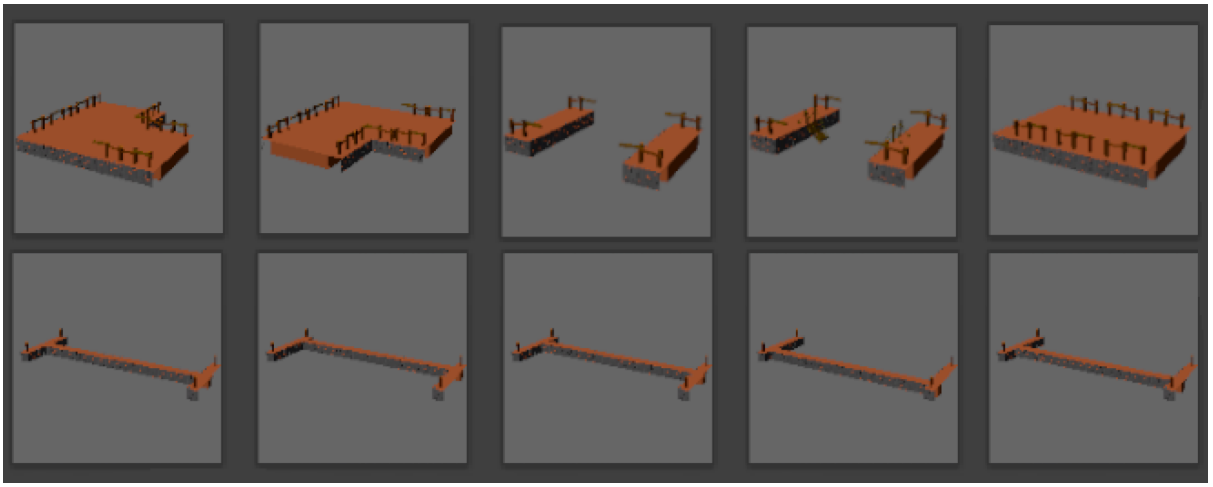
The camera is always positioned behind the character and centered on the path, the camera closely follows the movements of Tom, the main character, to maintain a consistent and coherent perspective.

When the character performs a slide on the ground, the camera has a specific animation that lowers it on the Y-axis, focusing on the character to allow the player to see how Tom slides under objects. This animation enhances the player's immersion by offering a detailed and exciting view of the action. Similarly, when Tom jumps, the camera tilts slightly backward, providing a sense of elevation and dynamism.

During turns, the camera simply follows the character from behind, ensuring the player always has a clear and direct view of the path ahead. These camera animations and movements not only make the gameplay experience more engaging but also allow for greater immersion by fluidly adapting to the character's actions, providing a visually rich and exciting gaming experience.

2.3.9. Procedural Map

The procedural map generation script in Unity dynamically creates a game environment by instantiating and positioning tile blocks based on predefined probabilities and rules. At its core, the script maintains lists of different types of tiles—initial blocks, straight blocks, and direction-changing blocks—each with associated probabilities that dictate their likelihood of being selected. During the game's initialization, the script generates an initial set of tiles and positions them appropriately. As the player progresses, new blocks are continuously generated ahead of the player, ensuring the map extends seamlessly. The script carefully manages the placement and rotation of each block to create a coherent path, while also deleting old blocks that are no longer needed to optimize performance. This approach allows for an infinite or large dynamically created map, providing a varied and engaging gameplay experience.



In addition to the core functionality, the script includes several key features to enhance the map generation process. The images show different types of platforms and paths that can be part of the procedural map. These platforms vary in shape and size, including straight paths, curves, and elevated sections that add complexity to the game environment. The script ensures that these elements are placed in a logical sequence, maintaining the flow of the game and presenting new challenges to the player.

One notable aspect of the script is its ability to adapt to the player's movements in real-time. As the player advances, the script dynamically adjusts the map, ensuring a continuous and smooth gameplay experience. This is achieved by calculating the player's position and generating new tiles ahead while removing those that are no longer needed. This not only optimizes performance but also keeps the game environment fresh and unpredictable.

The script also incorporates randomization within defined constraints, ensuring that each playthrough offers a unique experience. The placement of tiles is not entirely random but guided by probabilities and predefined rules that maintain the coherence of the map. For example, straight blocks might have a higher probability of being selected after a direction-changing block, ensuring that the map remains navigable.

In the game, the procedural generation of decorations, obstacles, and power-ups is managed through specialized scripts, each designed to enhance the gameplay experience by dynamically populating the environment with various elements.

The ObstacleSpawner script is responsible for generating different types of obstacles: destructible, killer, and roll obstacles. Each type has its own set of properties and behaviors, which are defined in the ObstacleData structure. This structure includes information such as the obstacle's prefab, its width, and whether it can rotate freely. During gameplay, the script selects an obstacle based on its type, determines a random position within a specified range, and instantiates it. If the obstacle is marked as rotate-free, it is assigned a random rotation, adding variability to its appearance. This dynamic obstacle generation ensures that the player encounters a variety of challenges, keeping the game engaging and unpredictable.

2.4. Screens Description

2.4.1. Menu

At the top center of the menu, the game title "Medieval Run" is displayed. The background features a cartoon-style landscape depicting either a sunset or sunrise, with soft, warm colors that create a cozy and serene atmosphere. Below the title, there are two navigation tabs: "Skills" and "Heroes". The "Skills" tab is currently selected, indicating that we are in the skills section of the game, while the "Heroes" tab is not selected.

The skills section shows three abilities that can be upgraded: "Luck", "Magnetic", and "Shield". Each of these abilities has a progress bar with four empty slots, indicating that no upgrades have been made yet. To the right of each skill, there is a yellow upgrade button with the text "Upgrade 300", signifying that upgrading each skill costs 300 coins.

In the top right corner of the screen, there are two counters. One shows the number of available coins, represented by a golden coin icon and the number "0" next to it. The other counter shows the number of trophies, represented by a trophy icon, also with the number "0" next to it.

At the bottom right of the menu, there are three action buttons. The "Credits" button allows you to view the game's credits. The "Save and Exit" button lets you save your progress and exit the menu. The largest button, "Play", lets you start playing the game. These elements are arranged clearly and accessibly, making it easy for the user to navigate and interact with the game.


MEDIEVAL
RUN
1150
3323...
SKILLS
HEROES
LUCK
MAGNETIC
SHIELD
UPGRADE
MAX
UPGRADE
MAX
UPGRADE
MAX
CREDITS
SAVE AND EXIT
PLAY

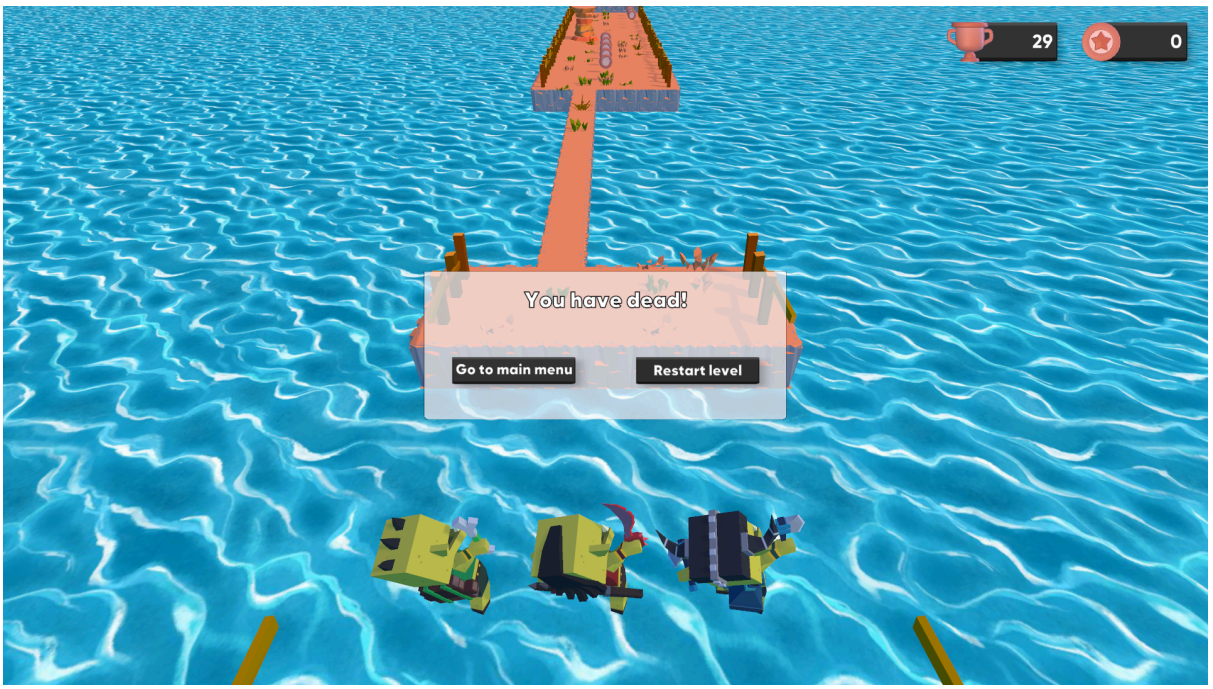

MEDIEVAL
RUN
1150
3323...
SKILLS
HEROES
LUCK
MAGNETIC
SHIELD
UPGRADE
MAX
UPGRADE
MAX
UPGRADE
MAX
CREDITS
SAVE AND EXIT
PLAY

2.4.2. Credits

The screen displayed is the credits screen for the game "Medieval Run," which acknowledges the contributions of both the producers and the creators of various assets used in the game. The producers, Héctor Núñez Carpio and Gissela Borrás Zaplana, are recognized for overseeing the game's development and ensuring the project stayed on track. The assets section credits the creators and studios that provided essential resources, including the Skybox Extended Shader by BOXOPHOBIC for background visual effects, the Low Poly Cartoon Item Pack by Lost Panda Games for game world objects, and the LowPoly Terrain Mesa by Fatty War for landscape features. Additionally, it acknowledges the Orcs Mega Pack by Meshtint Studio for character models, the RPG Tiny Hero Wave Polyart by Dungeon Mason for hero character models, and the Hyper Casual FX Pack Vol.1 by Kyeoms for special effects. This screen highlights the collaborative effort behind the game, offering transparency and appreciation for the diverse sources of its content.

2.4.3 Game Over

The screen displayed shows the "Game Over" scene for the game "Medieval Run." In this scene, the player has failed the level, which is indicated by the message "You have dead!" prominently displayed in the center of the screen. The backdrop features a water environment, suggesting that the player's character and their allies have fallen into the water, resulting in their demise. The screen provides two options for the player: "Go to main menu" and "Restart level." The first option allows the player to return to the game's main menu, where they can choose to start a new game or adjust settings. The second option lets the player restart the current level, giving them an immediate chance to try again and improve their performance. This scene effectively communicates the end of the current attempt and offers a clear and user-friendly way for the player to continue their gaming experience.


29
0
You have dead!
Go to main menu
Restart level

2.4.4. Instructions

The screen displayed is the instructions screen for the game "Medieval Run." This screen provides essential information to the player about the gameplay mechanics, controls, and various interactive elements within the game.

At the top of the screen, under the heading "Instructions," the screen is divided into three main sections: Destructible Objects, Controls, and Shortcuts.

Destructible Objects

On the left side, the screen shows images of different destructible objects that the player will encounter. These include various types of crates and cactus plants. The player can break these objects to clear their path and possibly reveal hidden items or power-ups.

Controls

In the center, the screen displays the directional controls for moving the player character. The arrow keys are shown with the corresponding actions:

- Up Arrow: Jump
- Down Arrow: Scroll (or possibly slide)
- Left Arrow: Move Left
- Right Arrow: Move Right

Additionally, there is a key for "Hit," represented by a spacebar-like icon, indicating the action to attack or break objects.

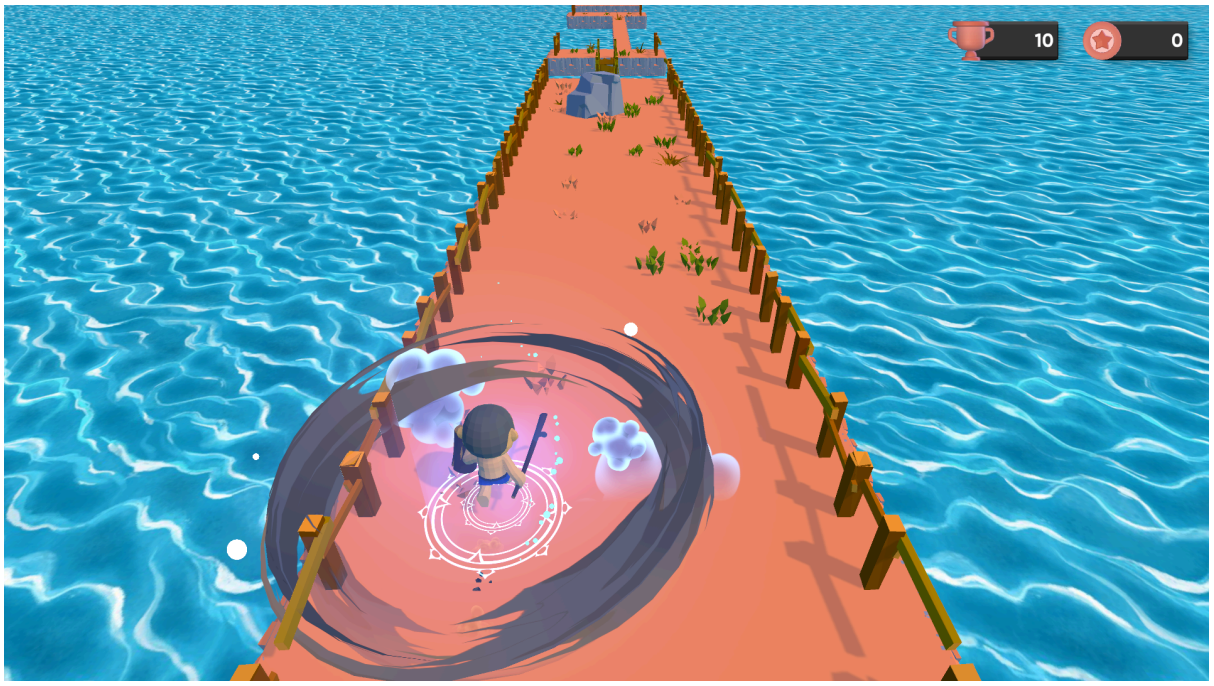
Shortcuts

On the right side, the screen lists shortcuts for activating various power-ups and actions:

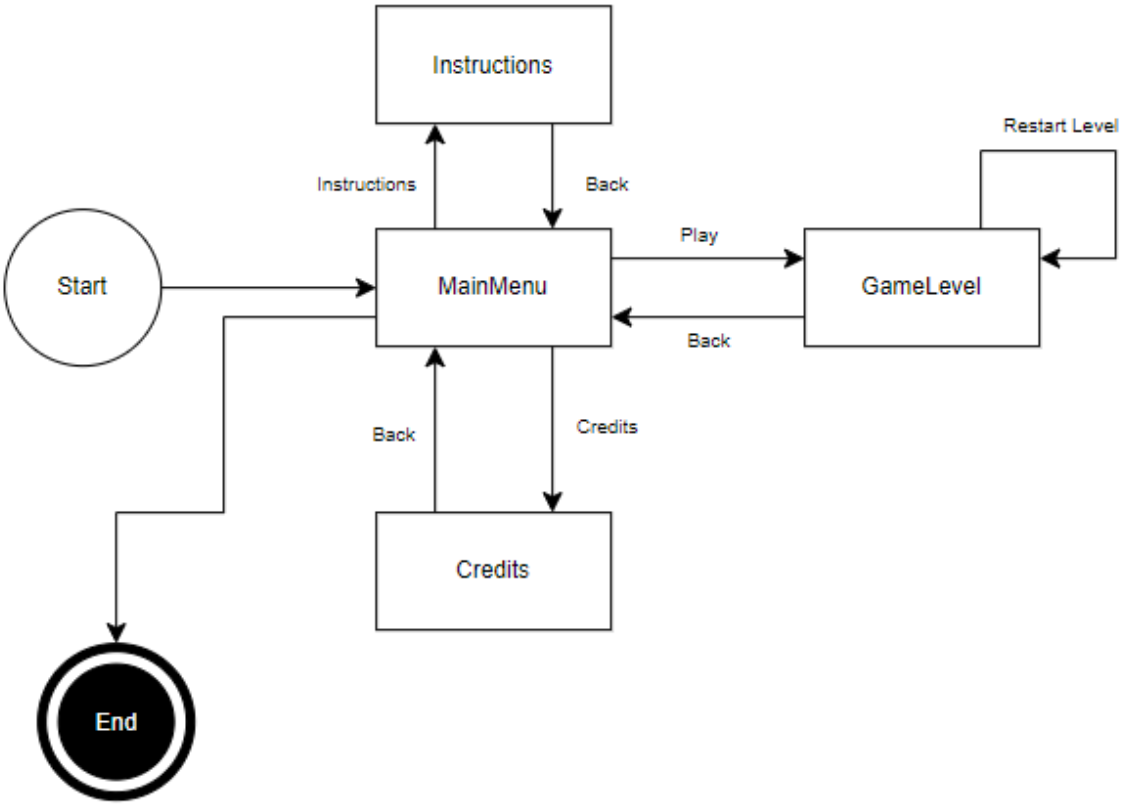
- W: Use the Magnet power-up, which attracts nearby coins.
- Q: Use the Shield power-up, which provides temporary protection.
- G: Activates a special ability, possibly related to a crown icon shown next to it.

This instructions screen is designed to quickly familiarize players with the basic controls and key gameplay elements, ensuring they understand how to navigate and interact within the game.

2.4.5. GameLevel



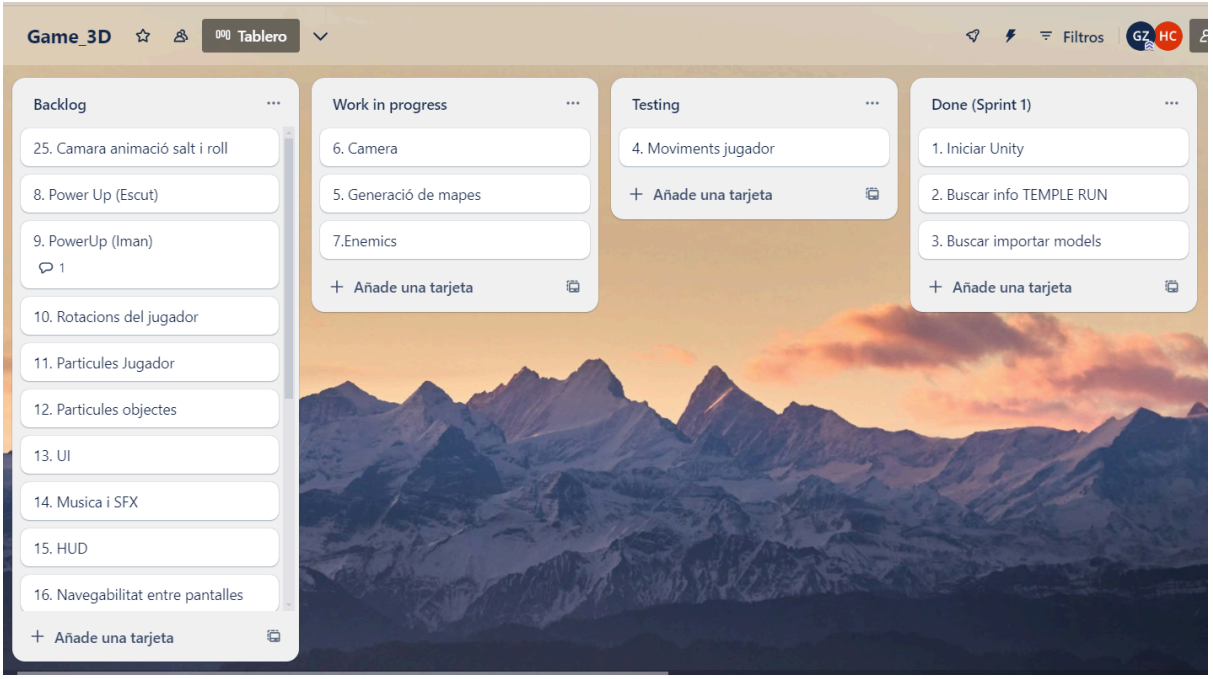
2.5. Screens Diagram



3. Methodology

3.1. Organization

We organized ourselves so that in class, we could collectively address any questions and problems that arose while working at home. This strategy allowed us to tackle and solve issues collaboratively, leveraging the different perspectives and skills of each team member. For version control, we used GitHub, where each person worked on their own branch. This allowed us to develop and test new features in isolation, ensuring that no errors were introduced into the main codebase. When we confirmed that there were no merge conflicts with the develop branch, we would push our changes, thus maintaining the project's integrity and stability.



Furthermore, we structured our project into four sprints, each lasting two weeks. This approach enabled us to set clear and manageable goals for each sprint, facilitating continuous and measurable progress. At the end of each sprint, we reviewed our achievements and planned the next steps, which helped us stay focused and adapt to any changes or challenges that arose. This methodology not only improved our efficiency but also provided greater clarity about the project's status and direction at all times.

3.2. Sprints

In the development of our 3D video game project, we adopted a structured approach by dividing the workload into four distinct sprints, each lasting two weeks. This methodology allowed us to set clear goals, manage tasks efficiently, and ensure continuous progress.

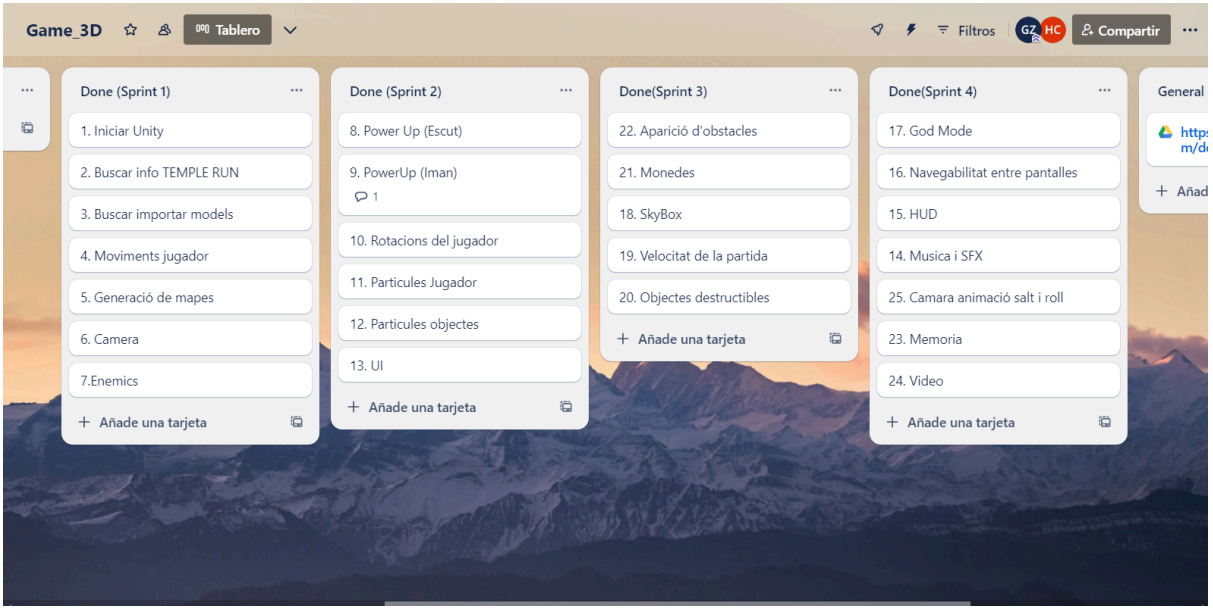
During Sprint 1, we focused on laying the groundwork for the project. We began by initializing Unity and gathering information about similar games, such as "Temple Run," to draw inspiration and understand best practices. Our tasks included initializing Unity, researching information about "Temple Run," importing models, implementing basic player controls and movements, developing the initial procedural map generation system, configuring the camera to follow the player, and designing and integrating enemy characters.

In Sprint 2, we built upon the foundations laid in Sprint 1 by adding new features and enhancing the gameplay experience. The key tasks included implementing the shield power-up to protect the player, adding the magnet power-up to attract coins, refining the player's rotation mechanics, creating particle effects for player actions, adding particle effects for interactions with objects, and designing and integrating the user interface elements.

Sprint 3 was dedicated to expanding the game's content and improving the overall gameplay experience. Our main objectives were introducing various obstacles that the player must avoid or interact with, adding collectible coins to the game, enhancing the visual environment with a dynamic skybox, adjusting the game's speed mechanics for better pacing, and implementing objects that the player can destroy.

In the final Sprint 4, we focused on polishing the game and adding finishing touches. The tasks included implementing a special mode for enhanced gameplay, ensuring smooth transitions between different game screens, finalizing the HUD for displaying game information, adding background music and sound effects to enhance the auditory experience, creating camera animations to follow the player during jumps and rolls, ensuring the game efficiently uses memory resources, and producing a promotional video to showcase the game.

By structuring our work into these sprints, we were able to maintain a focused and organized approach, allowing us to systematically build and refine our game, "Medieval Run." This iterative process facilitated continuous improvement and ensured that we met our development goals effectively.



3.3. Meetings

Sprint 1: Initial Setup and Foundation

During Sprint 1, our primary focus was on setting up the development environment and laying the foundation for the project. In our first meeting, we discussed the project scope, set up Unity, and assigned initial tasks. We also reviewed "Temple Run" to draw inspiration and establish best practices. Mid-sprint, we had a check-in meeting to address any challenges with Unity setup, model imports, and basic player movements. By the end of the sprint, we conducted a review meeting where we successfully implemented the initial map generation system, player movements, and basic camera setup. We identified areas for improvement and planned the tasks for Sprint 2.

Sprint 2: Feature Development and Enhancements

In Sprint 2, we focused on adding new features and enhancing the gameplay. Our initial meeting involved finalizing the shield and magnet power-up designs and assigning tasks for player rotations, particle effects, and UI development. Mid-sprint, we discussed the integration progress of these features, resolved any merge conflicts on GitHub, and ensured that everyone was on track. By the end of the sprint, we held a review meeting where we tested the new power-ups, refined player mechanics, and evaluated the UI elements. We gathered feedback and adjusted our plans for the next sprint.

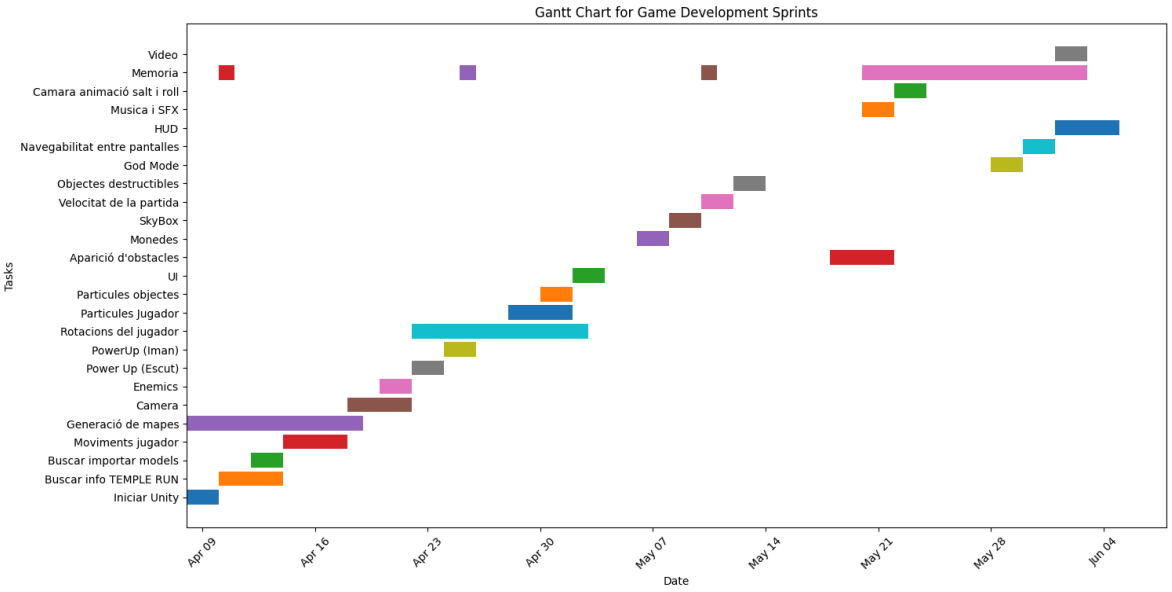
Sprint 3: Content Expansion and Gameplay Refinement

Sprint 3 was dedicated to expanding the game's content and refining the overall gameplay experience. In our initial meeting, we outlined the introduction of obstacles, collectible coins, and visual enhancements like the skybox. Mid-sprint, we checked in to monitor progress on these tasks, particularly focusing on game speed adjustments and the implementation of destructible objects. We also addressed any technical challenges encountered. At the end of the sprint, we reviewed the newly added obstacles and collectibles, assessed the game's pacing, and ensured that the visual elements were integrated seamlessly. We gathered input for further refinement in the final sprint.

Sprint 4: Polishing and Final Touches

In the final Sprint 4, our goal was to polish the game and add finishing touches. Our initial meeting focused on implementing the God Mode, ensuring screen navigation was smooth, and finalizing the HUD. We also planned the addition of background music, sound effects, and camera animations for jumps and rolls. Mid-sprint, we reviewed the memory usage and performance, ensuring that the game ran efficiently. We also began working on the promotional video. By the end of the sprint, our review meeting involved extensive testing of all game features, final adjustments based on feedback, and completing the promotional video. We concluded the sprint with a polished and ready-to-launch version of "Medieval Run."

3.4. Gantt Diagram



4. Conclusions

In this project, "Medieval Run," we worked as a team to create an endless runner game that is both fun and challenging. Throughout the development process, we learned and applied various technical skills and design principles.

We organized our work into sprints, which allowed us to break the project into manageable parts and make systematic progress. Each sprint had clear objectives, helping us focus on specific tasks and measure our progress. This structure enabled us to identify and solve problems quickly, keeping the project on track.

Collaboration was key to the success of our project. Each team member brought their unique skills to the table, and we worked together to overcome challenges. We used tools like GitHub for version control, allowing us to work in parallel without interfering with each other's work. Regular meetings were crucial for discussing progress, resolving issues, and planning the next steps.

The game development included implementing various technical elements, such as player control, enemies, obstacles, and procedural map generation. We learned to create and manage these elements using Unity, providing valuable hands-on experience in game programming.

Besides the technical aspects, we also paid attention to design and gameplay. We implemented a HUD system to provide the player with essential information during the game and designed power-ups that add an extra layer of strategy and fun. The procedural map generation ensures that each game session is unique, maintaining the player's interest.

In conclusion, this project has been a significant learning experience. We have not only improved our technical skills and project management abilities but also learned to work effectively as a team. The final product is a game that reflects our effort and dedication, and we are very proud of it. This experience has better prepared us for future projects in game development and beyond.

5. Bibliography

- BOXOPHOBIC. (n.d.). Skybox Extended Shader. Retrieved from BOXOPHOBIC.
- Lost Panda Games. (n.d.). Low Poly Cartoon Item Pack. Retrieved from Lost Panda Games.
- Fatty War. (n.d.). LowPoly Terrain Mesa. Retrieved from Fatty War.
- Meshtint Studio. (n.d.). Orcs Mega Pack. Retrieved from Meshtint Studio.
- Dungeon Mason. (n.d.). RPG Tiny Hero Wave Polyart. Retrieved from Dungeon Mason.
- Kyeoms. (n.d.). Hyper Casual FX Pack Vol.1. Retrieved from Kyeoms.
- Shepherd, K., & Luckyanova, N. (2011). Temple Run [Video game]. Imangi Studios.
- Geig, M., & Henley, S. (2021). *Unity 2021 Cookbook: Over 140 Recipes to Take Your Unity Game Development Skills to the Next Level, 4th Edition*. Packt Publishing.
- Wikipedia. (n.d.). Temple Run. Retrieved from https://en.wikipedia.org/wiki/Temple_Run