UNIVERSITY OF ECONOMICS AND HUMAN SCIENCES IN WARSAW

THE FACULTY OF INFORMATICS

Burak Kasan

Full-time Study

Index Number 35873

Mobile Game Development: Hyper-casual Game

Documentation for the diploma project

prepared under your guidance Dr. Ruslana Prus

Warsaw, 2023

Contents

ABSTRACT

The main goal of the project is to create a Hyper-casual game. The aim of the developer was to get information by examining similar games and applications that might be useful for their game. To develop the game, Unity was used. Reasons behind choosing to develop a hyper-casual game includes two main thoughts. Firstly, these games are easier to build and use less resources than more complex games. This allows creators to create and release new games more quickly, boosting the likelihood of a hit game. Second, because of their simple gameplay and ease of use, hyper-casual games appeal to a wide range of people, including casual gamers and non-gamers. Its broad appeal broadens the potential audience for these games, perhaps increasing downloads and revenue.

Finally, the project's goal is to create a Hyper-casual game called 'Burger Master,' which belongs to the type of mobile games distinguished by uncomplicated concepts, short play durations, and simple gameplay. And the result of this work is a playable alpha version that includes 7 levels.

STRESZCZENIE

Głównym celem projektu jest stworzenie gry Hyper-casual. Celem autora było uzyskanie informacji poprzez zbadanie podobnych gier i aplikacji, które mogą być przydatne w ich grze. Do opracowania gry wykorzystano Unity. Powody, dla których zdecydowaliśmy się stworzyć hiper-casualową grę, obejmują dwie główne myśli. Po pierwsze, te gry są łatwiejsze do zbudowania i zużywają mniej zasobów niż bardziej złożone gry. Pozwala to twórcom na szybsze tworzenie i wydawanie nowych gier, zwiększając prawdopodobieństwo przeboju. Po drugie, ze względu na prostą rozgrywkę i łatwość użytkowania, gry hiper-casualowe przemawiają do szerokiego grona osób, w tym zwykłych graczy i niegraczy. Jego szerokie zainteresowanie poszerza potencjalną publiczność tych gier, być może zwiększając liczbę pobrań i przychody.

Wreszcie, celem projektu jest stworzenie hiper-casualowej gry o nazwie „Burger Master”, która należy do rodzaju gier mobilnych wyróżniających się nieskomplikowaną koncepcją, krótkim czasem gry i prostą rozgrywką. Efektem tej pracy jest grywalna wersja alfa zawierająca 7 poziomów.

1.COMPETITION ANALYSIS FOR HYPER-CASUAL GAMES

Traditional mobile games, mid-core games and hardcore games are competing hyper-casual gaming responses. Each of these solutions has advantages and disadvantages.

1.1 TRADITIONAL GAMES

Traditional mobile games are more complicated than hyper-casual games and frequently feature more realistic storylines and graphics. These games appeal to players that want an enhanced gameplay experience and are ready to put more effort to understanding the game's rules and mechanics. Traditional mobile games, on the other hand, are frequently more expensive to develop and demand more resources than hyper-casual games. This can make it more difficult for developers to create and release new games on a timely basis, perhaps limiting their chances of finding a hit game. Following images displays two of the hit Traditional Games:

Candy Crush

Clash of Clans

1.2 MID-CORE GAMES

Mid-core games stand between casual and hardcore. These games provide more complex gameplay than hyper-casual games while maintaining a low entry price. They often require more strategy and skill than hyper-casual games and may have longer play sessions. Mid-core games, on the other hand, can be more difficult than hyper-casual games, which may restrict their appeal to some players. Hearthstone and Clash Royale can be given as examples for this genre of Mobile Game Development.

Clash Royale

Hearthstone

1.3 HARDCORE GAMES

Hardcore games are the most complex and interesting sort of mobile game. These games frequently have sophisticated stories, graphics, and gameplay mechanisms. They necessitate a tremendous time and effort investment from players, who must understand the complexities of the game in order to succeed. For players searching for a deep, immersive gaming experience, hardcore games can be incredibly interesting and enjoyable. Hardcore games, on the opposite side, are frequently more expensive to design and demand more resources than other sorts of mobile games. This can make it more difficult for developers to create and release new games on a timely basis, perhaps limiting their chances of finding a hit game. Fortnite and PUBG (PlayerUnknown's Battle Grounds) are two examples of this genre of the Mobile Game Environment.

PUBG

Fortnite

Finally, the solution used is determined by the developer's goals and the player's preferences. Because of their low development costs and high revenue potential, hyper-casual games are appealing to creators, whereas standard mobile games, mid-core games, and hardcore games offer more complicated gaming experiences but may require more resources and investment.

2. USED TECHNOLOGIES THROUHGOUT THE PROJECT

2.1 UNITY

Unity is a cross-platform game engine that is extensively used for developing interactive 2D and 3D applications in a number of industries including such film, automobile, architecture, and engineering. It provides game developers with a comprehensive set of tools and resources for developing and publishing games on numerous platforms such as Windows, macOS, iOS, Android, and consoles such as Xbox and PlayStation.

Reasons of choosing Unity vary since Unity has several advantages for producing hyper-casual games. Let me state them below:

- User-friendly interface: Unity has a user-friendly interface that allows developers, even those with no prior game production knowledge, to quickly create and prototype super casual games.

- Unity's huge library of pre-built components, plugins, and tools enables rapid prototyping and development of super casual games. This allows developers to iterate quickly and bring their games to market more quickly.

- Cross-platform compatibility: Unity enables creators to make ultra-casual games that can operate on a variety of platforms, including mobile devices, consoles, and desktop computers, with no effort.

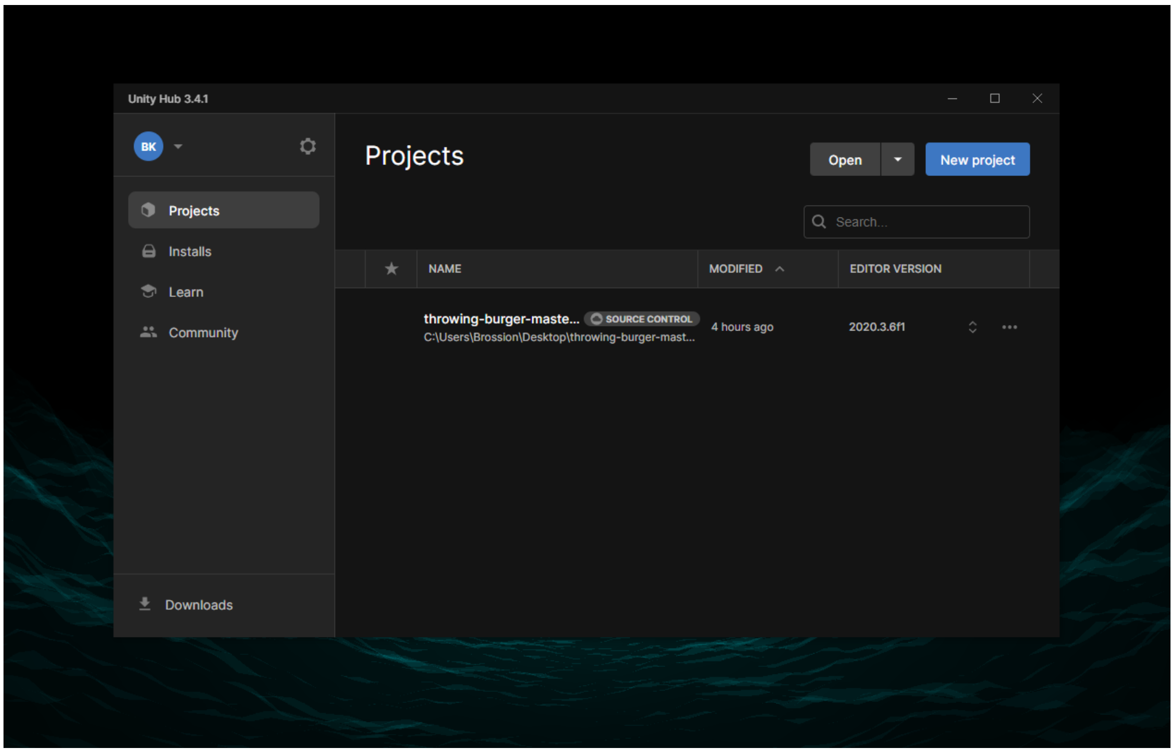
- Unity provides a free version of its technology that may be used to create and publish hyper-casual games. This low entry price is especially appealing to indie developers and small teams with limited resources.

- Unity has a big and active developer and user community that shares resources, offers advise, and provides support to one another. For hyper casual game makers trying to learn and progress, this community might be a wonderful resource.

Altogether, Unity's accessibility, ease of use, and versatility make it a popular choice for making hyper casual games, and its benefits can assist creators in creating compelling and successful games in this genre.

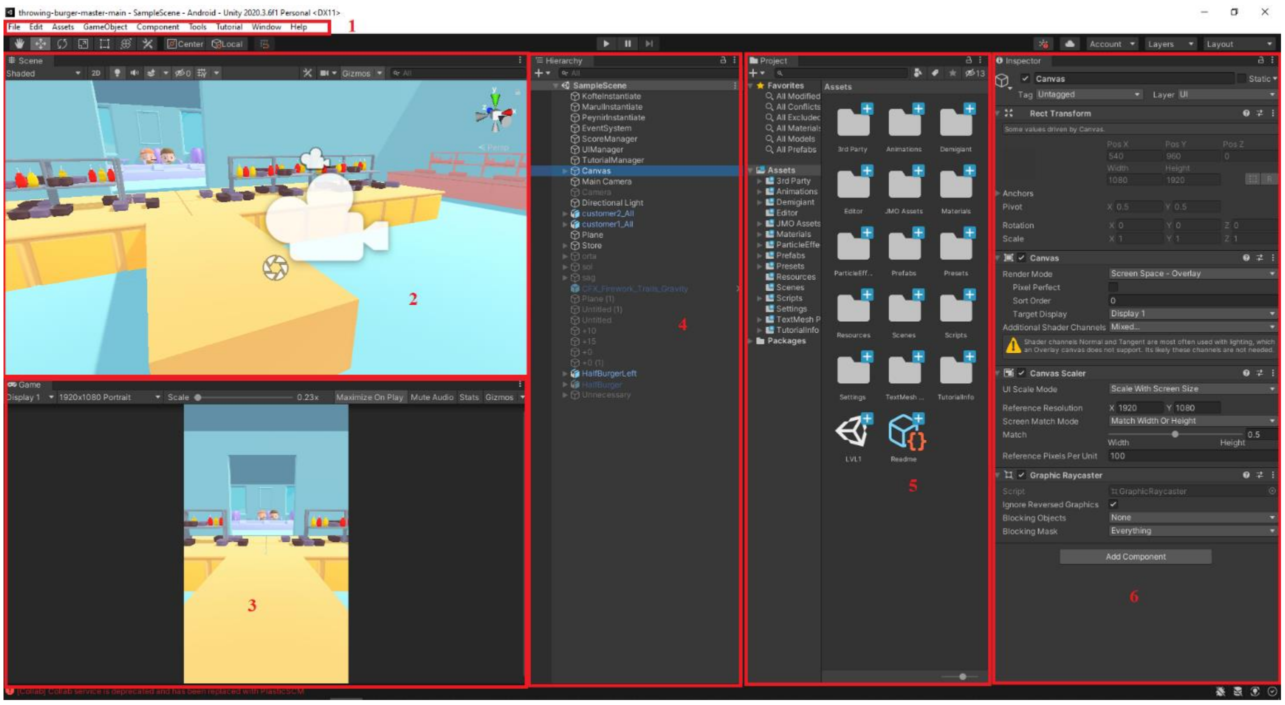
2.2 BASICS OF UNITY

When you access Unity Hub, you will see user-created projects. After that, double-click the folder. The Unity interface will then display on the window. There, can be created any folders or games as many possible as you can in here.



The Unity interface includes 6 panels:

1. Toolbar: Different buttons for interacting with your project
2. Scene Panel: It delivers the playground's 2D and 3D vision. (Scene)
3. Game Panel: It makes it available for you to preview your game.
4. Hierarchy Panel: This is a panel where all of the items in the scene are listed. Objects are referred to as "Game Objects" in Unity.
5. Project Panel: All in-game sources can be located here. For example, 3D and 2D objects, photos, sound files, typefaces, and so on.
6. Inspector Panel: This panel contains detailed information on the selected object. As in the image below, it is Canvas.



Canvas
Tag Untagged Layer UI

Rect Transform
Some values driven by Canvas.
Pos X 540 Pos Y 960 Pos Z 0
Width 1080 Height 1920
Pivot X 0.5 Y 0.5
Rotation X 0 Y 0 Z 0
Scale X 1 Y 1 Z 1

Canvas
Render Mode: Screen Space - Overlay
Pixel Perfect
Sort Order: 0
Target Display: Display 1
Additional Shader Channels: Mixed...
Shader channels Normal and Tangent are most often used with lighting, which an Overlay canvas does not support. Its likely these channels are not needed.

Canvas Scaler
UI Scale Mode: Scale With Screen Size
Reference Resolution: X 1920 Y 1080
Screen Match Mode: Match Width Or Height
Match: 0.5
Reference Pixels Per Unit: 100

Graphic Raycaster
Script: GraphicRaycaster
Ignore Reversed Graphics: ✓
Blocking Objects: None
Blocking Mask: Everything

2.3 VISUAL STUDIO CODE

As default C# compiler, Unity choses Visual Studio Code. But VSC hasn't been installed at the beginning of the project. After progressing on the project a while, VSC seemed more comfortable and have lots of options that makes it easier for developers such as 'Unity Code Snippets'. Developer switched to VSC upon that thought.

For the developer, reasons why Visual Studio Code is an excellent code editor while developing any game with Unity are:

- Cross-platform support: Because Visual Studio Code is accessible on Windows, macOS, and Linux, it is an excellent alternative for developers who work on multiple platforms.
- C# support: C# is the major programming language used to create Unity games. With tools like IntelliSense, code navigation, and debugging, Visual Studio Code provides outstanding support for C# development.
- Lightweight and customizable: Visual Studio Code is lightweight and customizable, so you can tailor it to your specific requirements. You can add additional features, modify the theme and syntax highlighting, and create your own keyboard shortcuts by installing extensions.

- Integrated debugging: Debugging Unity games is built into Visual Studio Code, allowing you to set breakpoints, step through code, and inspect variables while the game is running.
- Source control integration: Git, a popular version control system used by many developers, is embedded into Visual Studio Code. This simplifies project management and collaboration with other developers.

Generally, Visual Studio Code is an extremely powerful and adaptable code editor that is ideal for Unity game development. It can help you streamline your workflow, enhance productivity, and make writing, debugging, and managing code easier.

3. DEVELOPING PROCESS OF THE PROJECT

Developing this game was entertaining and compelling for the developer. Since he didn’t have any mentor to lead him about what has to be done, lots of obstacles has been faced on the way to finish this project. Enormous amount of researching has been done and a lot of digging in code example sites to check what’s done in what way. After completing the re-searching, sketch of my game has been made. All the steps have been acquired to develop the game. Here is the development process:

1. Ideation
2. Prototyping
3. Testing
4. Iteration
5. Launch
6. Analytics and optimization

3.1 IDEATION

Creating a unique and interesting game concept is the first stage in designing a hyper-casual game. This can be accomplished through brainstorming concepts, researching popular games in the genre, or discovering a market need that the game could fill. And developer did all the research. This required a lot of gameplays since he had to see the mechanics of other hyper-casual games. Upon playing abundant amount of hit hyper-casual games he has decided to make a 3D Burger Throwing Game and due to the environment he was in at that moment and he decided to name it: “Burger Master”.

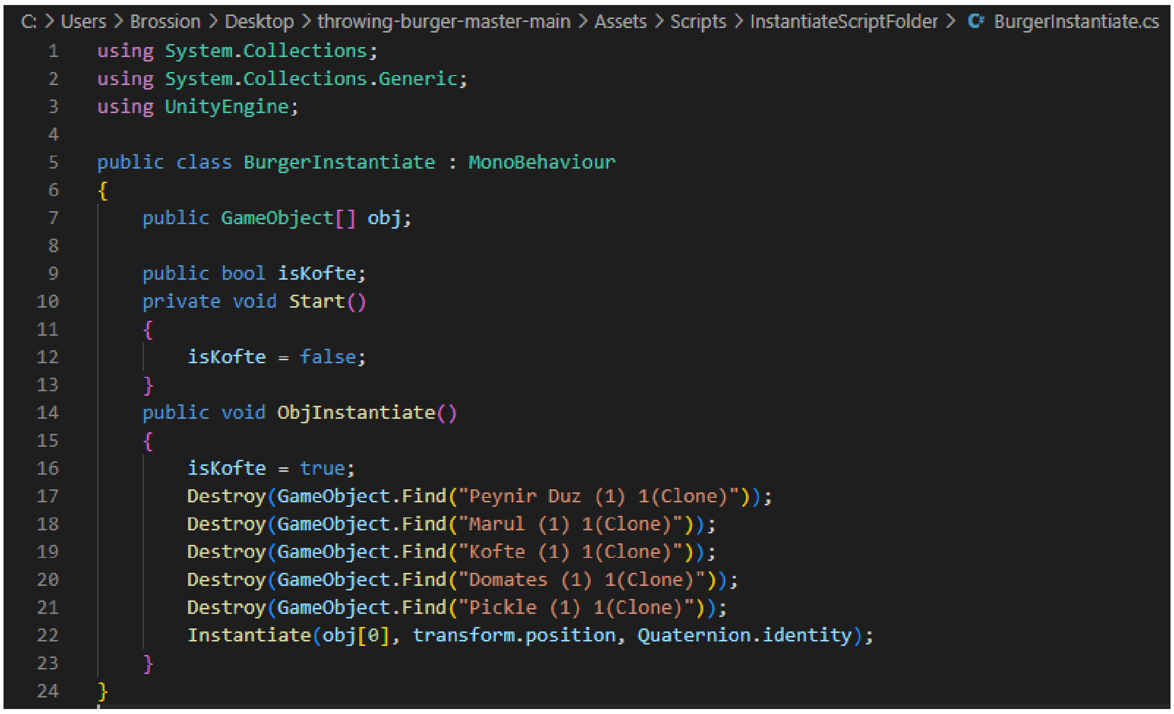
3.2 PROTOTYPING

Whenever you have an idea for a game, you should build a prototype to test the fundamental mechanics and gameplay. This might be a basic version of the game that can be played on a mobile device or computer. Usually, a hyper-casual game developer creates a prototype in one week and then present it to the publisher to gain acknowledge about their opinion. A presentation of the finished project’s scripts can be found in this section:

3.2.1 Instantiating Objects Scripts

In hyper-casual games age range of players are quite huge. 5-year-old to 50, sometimes more, year old people can access to these games and play. This is because game mechanics are easy to learn and apply. Therefore, for players to adapt the mechanics easily, instantiating the burger materials by pressing buttons option seemed more easy comparing other mechanics. When player presses a button, the material shown on the image will be instantiated and Player will be able to throw it in order to prepare the order.

Player is able to instantiate the GameObject by calling the Instantiate() method in a script. Here is the implementation in one of the Instantiate scripts.



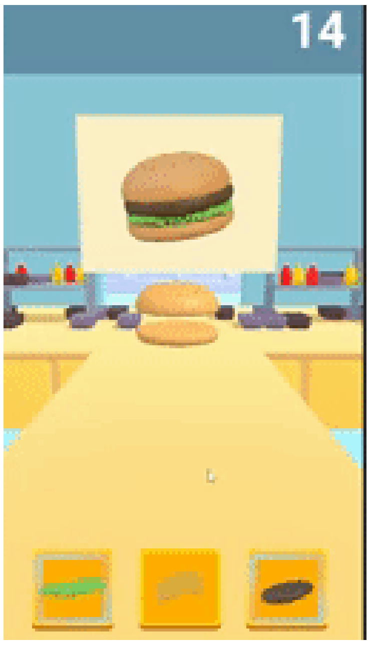
C: > Users > Brossion > Desktop > throwing-burger-master-main > Assets > Scripts > InstantiateScriptFolder > BurgerInstantiate.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class BurgerInstantiate : MonoBehaviour
{
    public GameObject[] obj;

    public bool isKofte;
    private void Start()
    {
        isKofte = false;
    }
    public void ObjInstantiate()
    {
        isKofte = true;
        Destroy(GameObject.Find("Peynir Duz (1) 1(Clone)"));
        Destroy(GameObject.Find("Marul (1) 1(Clone)"));
        Destroy(GameObject.Find("Kofte (1) 1(Clone)"));
        Destroy(GameObject.Find("Domates (1) 1(Clone)"));
        Destroy(GameObject.Find("Pickle (1) 1(Clone)"));
        Instantiate(obj[0], transform.position, Quaternion.identity);
    }
}
```

Since throwable object is a child object, a public Game Object array is added and Prefabs that have already been created had inserted into the array. This script is used to instantiate the meat for the burger. If it’s pressed the Boolean value ‘isKofte’ will be true and Meat will be chosen amongst other materials of the Burger.

3.2.2 Throwing Mechanism Scripts

In order to throw the chosen GameObject, Player needs to swipe the screen.



The code defines three Vector3 variables to store the ball's initial location, ending position, and force.

```
private Vector3 startPos, endPos, force; // touch start position, touch end
position, force
```

It also defines a float variable called factor for adjusting the swipe's strength. A Rigidbody component attached to the same game object is also mentioned in the script. The script creates a reference to a UIManager object in the Start() method, which is in charge of managing the game's UI elements. The script checks if the game has started and if the player has touched the screen in the Update() method. When the player touches the screen, the script saves the touch start location and time, translates the touch position from screen to world coordinates, and changes touchStarted to true.

```
if (Input.GetMouseButtonDown(0) && touchStarted == false /*&& uiMan-
ager.canTouch == true*/)
            {
                if (/*EventSystem.current.IsPointerOverGameObject() &&*/
IsPointerOverUIObject())
                {
                        Debug.Log("UI Button touchedDown!");
                        return;
                    }
                    // getting touch position and marking time when you touch
the screen
                    startTime = Time.time;
                    startPos = Input.mousePosition;
                    startPos.z = transform.position.z - Camera.main.trans-
form.position.z;
                    startPos = Camera.main.ScreenToWorldPoint(startPos);
                    touchStarted = true;
                    Debug.Log("Swiping has began.");
                }
```

When the player releases the touch, the script saves the end point of the touch, translates it from screen coordinates to world coordinates, calculates the swipe direction and distance, applies a force to the ball based on the swipe direction and distance, and sets touchStarted to false.

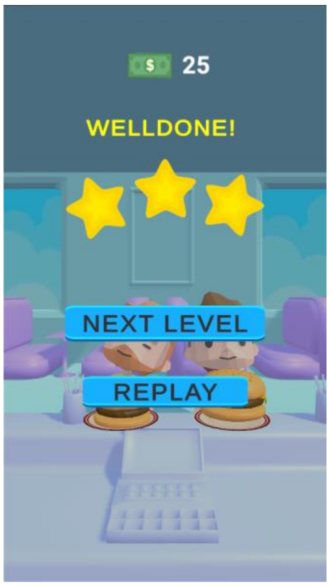
```
if (Input.GetMouseButtonUp(0) && touchStarted == true /*&& uiManager.can-
Touch == true*/)
                {
                    if (/*EventSystem.current.IsPointerOverGameObject() &&*/
IsPointerOverUIObject())
                    {
                        Debug.Log("UI Button touchedUp!");
                        return;
                    }
                    // getting release finger position
                    endPos = Input.mousePosition;
                    endPos.z = transform.position.z - Camera.main.transform.po-
sition.z;
                    endPos = Camera.main.ScreenToWorldPoint(endPos);

                    // calculating swipe direction in 2D space
                    force = (endPos - startPos).normalized;
                    force.z = force.magnitude;
                    //force /= (Time.time - startTime);

                    // add force to balls rigidbody in 3D space depending on
swipe time, direction and throw forces
                    if (Vector3.Distance(startPos, endPos) > 0.4f)
                    {
                        rb.AddForce(force * factor);
                    }
                    touchStarted = false;
                Debug.Log("Swiping has finished.");
                Debug.Log("force is: " + force + ", startTime is: " +
startTime + "startPos is: " + startPos + ", endPos is: " + endPos + ", fac-
tor is: " + factor);
            }
```

The script additionally checks to see if the player is touching a UI element and, if so, avoids the swipe.

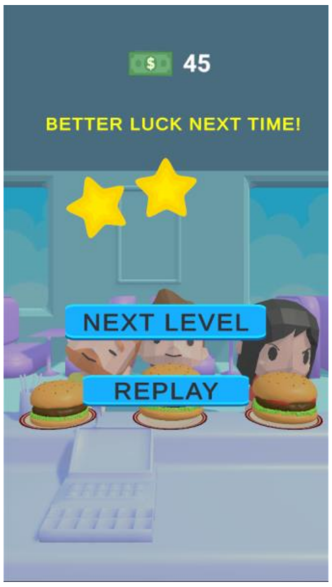
```
private bool IsPointerOverUIObject()
    {
        PointerEventData eventDataCurrentPosition = new PointerEvent-
Data(EventSystem.current);
        eventDataCurrentPosition.position = new Vector3(Input.mousePosi-
tion.x, Input.mousePosition.y, Input.mousePosition.z);
        List<RaycastResult> results = new List<RaycastResult>();
        EventSystem.current.RaycastAll(eventDataCurrentPosition, results);
        return results.Count > 0;
    }
```

3.2.3 Result Mechanism Scripts

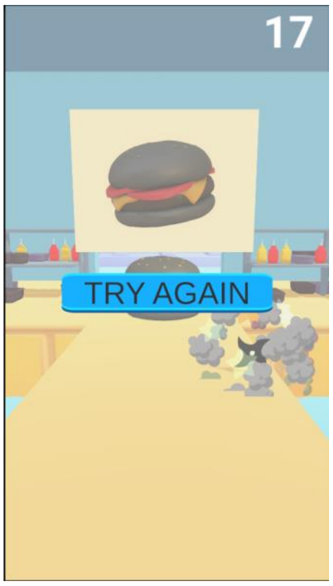
For resulting mechanism, three options are available for Player to achieve; ‘Victory’, ‘Better Luck Next Time’ and ‘Fail’. Victory is when user successfully prepare the Burgers.



Better Luck Next Time is when user fails to complete at least one of the orders.



Fail is when user fails to complete all orders or throws the objects out of Burger. Player also fails if same object is thrown twice.



In order to detect which result is taken by the player, a score variable is added to the scripts. Player can skip next level if at least one object is correct. Star system is added according to that. Here is a code example of how the score is calculated.

```
if (totalScore < 33)
        {
            Debug.Log("Try again!");
            yield return new WaitForSeconds(0.3f);
            tryAgainPanel.SetActive(true);
            generalScript.gameOver = true;
            uiManager.isGameStarted = true;
        }
        else if (totalScore >= 33 && totalScore < 34){…
```

Scores are calculated by collision. By Collision, it is meant if object collides with bread. When it does, some amount of point is added to the score. You can see an example down below:

```
if (other.gameObject.tag == "peynir")
        {
            peynirMiktari += 1;
            if (peynirMiktari == 1)
            {
                uiManager.isBad = false;
                if (score > 0)
                {
                    score -= scoreIncrease;
                }
            }
            else
            {
                uiManager.isBad = true;
                StartCoroutine(TryAgain());
            }
            Destroy(other.gameObject);
            cheese.SetActive(true);
            uiManager.canTouch = false;
            print(score);
            NextBTN();
        }
```

‘scoreIncrease’ variable adds or extracts score.

3.2.4 Assets and Animations

Finding the proper assets and post process was huge part of the project. Decent lighting should be added in order to have a good looking hyper-casual game.

Animations are downloaded from Mixamo.com and applied in Unity’s integrated animation panel. Mixamo.com is a website that provides a variety of 3D assets for use in video game development and other digital media projects, such as characters, animations, and other tools. The website provides a large collection of pre-made characters and animations that are compatible with a variety of 3D software programs and game engines. The automatic rigging technique in Mixamo makes it simple to quickly construct bespoke characters with minimal effort.

The process of improving the visual quality of a picture or video after it has been generated is referred to as post-processing. Color saturation, brightness, contrast, and other variables can be adjusted to achieve a specific look or style. Post-processing can also include the addition of special effects such as lens flares, depth of field, and motion blur to enhance the visual appeal of a picture or video.

Unity assets are 3D models, animations, scripts, and other materials that can be used to build interactive 3D experiences in the Unity game engine. These assets can be generated by developers, acquired from online marketplaces, or downloaded for free from free asset libraries. Unity assets can be used to make a wide range of games, including simple 2D puzzle games and complicated 3D multiplayer games. The Unity Asset Store is a popular marketplace for buying and downloading Unity assets, including Mixamo elements.

3.2.5. Results of Prototyping

After lots of coding, implementing and hard work, a playable prototype of the game that includes the gameplay mechanics and result mechanics was ready to test.

3.3 TESTING

After developing a prototype, it's critical to put it through its paces with a group of beta testers to gather input on the gameplay, mechanics, and overall experience. This feedback can be used to improve the game and make it more enjoyable. Since this is a one team project, mostly developer tested the game by himself but User feedback is important for every game development process so collaboration with testers was needed. Some of developer's friends had not much experience with hyper-casual games and some of them had professional experience due to their Functionality Quality Assurance (Game Tester) profile.

As a game developer I took advantage of debugging manually by 'Debug.Log'.

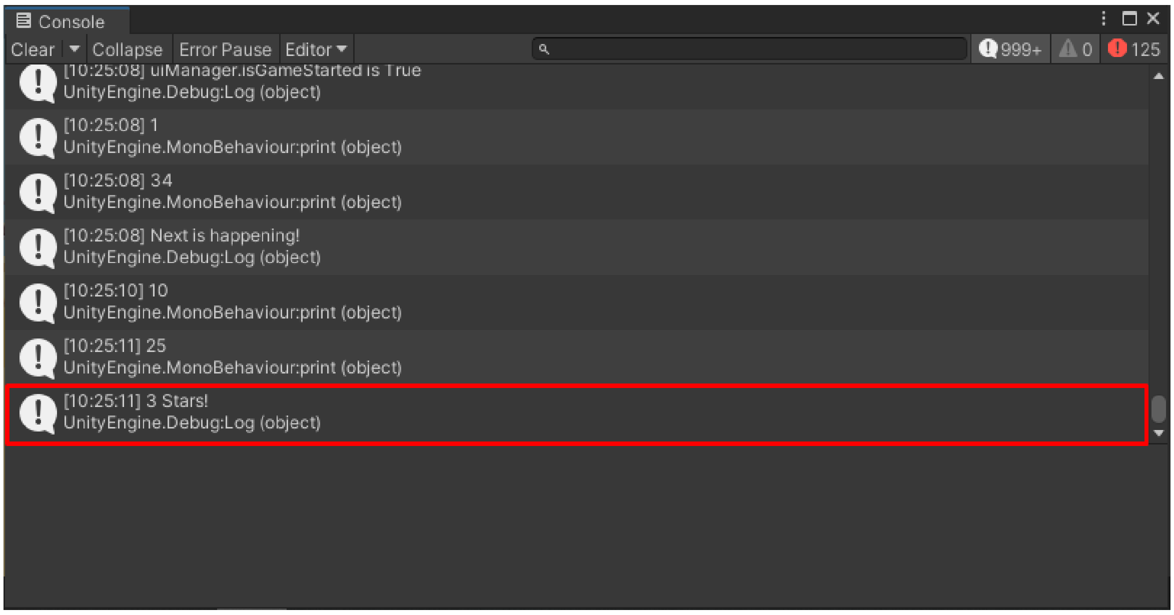
3.3.1 Usage of Debug.Log

The "debug.log" function in Unity is a built-in way for printing messages to the console for debugging. When you use the "debug.log" function in your script, Unity displays the provided message in the Unity Editor's Console window.

The "debug.log" function can be used to print to the console various types of information, such as the value of a variable, the status of a condition, or any other information you want to monitor while your game is running. You can use this method to debug your code and detect issues, monitor your game's performance, or simply keep track of what's going on in your game at runtime.

Here is how 'Debug.Log' is implemented:

```
…else if (totalScore == 67)
        {
            Debug.Log("3 Stars!");
            uiManager.isGameStarted = false;
            yield return new WaitForSeconds(0.3f);…
        }…
```



```
[10:25:08] uiManager.isGameStarted is True
UnityEngine.Debug:Log (object)
[10:25:08] 1
UnityEngine.MonoBehaviour:print (object)
[10:25:08] 34
UnityEngine.MonoBehaviour:print (object)
[10:25:08] Next is happening!
UnityEngine.Debug:Log (object)
[10:25:10] 10
UnityEngine.MonoBehaviour:print (object)
[10:25:11] 25
UnityEngine.MonoBehaviour:print (object)
[10:25:11] 3 Stars!
UnityEngine.Debug:Log (object)
```

3.4 ITERATION

It is critical to iterate and polish the game mechanics, graphics, and general user experience based on feedback from beta testers. Many rounds of testing and iteration may be required until the game is polished and ready for release. Since this game is only for diploma project, Developer doesn’t have plans to release it yet. But Iteration is an important part of the process so He iterated the game according to the feedback from the User feedbacks and added UI elements, Particle effects to make it more entertaining.

3.5 LAUNCH

Once the game is complete, it must be published on the app store or other platforms. This could include marketing and promotional activities to promote interest in the game and attract users. Launching for this project will be submitting it for diploma project. Although There is no intention of launching this game in hyper-casual game platforms such as Android and iOS, research has been done and abundant amount of knowledge has learnt about the process.

3.6 ANALYTICS AND OPTIMIZATION

Upon launch, it is critical to evaluate user engagement and performance data in order to find areas for improvement and optimization. This could include producing game updates or offering fresh content to keep consumers interested.

4. CONCLUSION

In conclusion, Developer has realized that there are lots of new things to discover, learn and apply to his developer profile. As a Gamer, he loves playing games and now that he has seen that creating one is as fun as playing itself, He is determined to follow this trail to the end even if he falls during it. He metaphorically fell too many times during the process of developing this game but he adjusted his mental according to that. Developer's motto is "You fall 9 times, but always get up for 10!".

Various things have been concluded along the project:

1. Knowledge of the hyper-casual genre: Developing a hyper-casual game can help the developer gain a greater understanding of the hyper-casual genre, its audience, and the game mechanics that work best for this type of game.

2. Iterative development strategy: An iterative development approach is frequently used in hyper-casual game creation, in which the developer produces a rudimentary game prototype, tests it with users, then iteratively refines the game depending on user feedback. The developer can determine whether this strategy was effective for their individual game at the end of the production process.

3. The importance of data analysis: The development of hyper-casual games is strongly reliant on data analysis, such as user acquisition, retention, engagement, and revenue statistics. The developer can determine whether their data analysis approach was effective in increasing the game's performance at the end of the development phase.

4. The importance of rapid prototyping: Hyper-casual games are often developed in a short period of time, and rapid prototyping is an important aspect of the process. The

developer can infer the effectiveness of their quick prototyping strategy and determine whether it worked effectively for their specific game.

Ultimately, at the end of the hyper-casual game development process, Developer learned that developers can draw conclusions about the development process and apply these insights to better future projects.

BIBLIOGRAPHY

- Dean Takahashi (2019), "The rise of hyper-casual games" - VentureBeat
- "Is Hyper Casual Gaming the Future of the Gaming Industry?" - https://www.amrita.edu/ahead/blog/is-hyper-casual-gaming-the-future-of-the-gaming-industry
- Ivan Fedyanin (May 10, 2021), “How to start hypercasual games production from scratch” - https://www.gamesindustry.biz/how-to-start-hyper-casual-games-production-from-scratch
- Unity documentation on using an external script editor: https://docs.unity3d.com/Manual/VisualStudioIntegration.html
- Visual Studio Code documentation on debugging: https://code.visualstudio.com/docs/editor/debugging
- Unity Scripting API: Debug.Log - https://docs.unity3d.com/ScriptReference/Debug.Log.html