Hi I’m James and this is buzzard

a space dogfighting game published to the google play store and itch io.

Players accumulate points by shooting down enemy ships, points that they can then use to purchase upgrades for their ship. Players are able to customise their ships engine, armour, energy system, heavy weapons and light weapons resulting in over 4 and a half trillion unique configurations for your ship.

Maybe one of these configurations will be good enough to beat the extra tough enemy frigate on round 5

Lets talk about what mobile features buzzard takes advantage of.

Gyroscope, by default the gyroscope is used to control the pitch and roll of the ship. This leaves your fingers free to use the boost, heavy weapon shoot and self destruct buttons.

In the settings menu the user is able to choose to not use the gyro and instead use an on screen virtual joystick.

The game uses the microphone to immerse the user into the world of buzzard. When the game first starts an interactive cutscene is played, the player must respond correctly to advance. The voice to text recognition is done using the hugging face api for unity.

When the player dies they see this cutscene. If the user navigates to the audio menu they are able to replace the sound with their own custom sound instead.

Finally vibration is used to give feedback both when navigating the ui of the game but also to immerse the player in gameplay. Light vibration is played when ui elements are selected. And when playing vibration is played when an enemy ship is destroyed. The strength of the vibration is linked to the distance from the enemy ship. Far away ships will cause the phone to emit a light vibration, medium range ships a medium vibration and close range ships a heavy vibration.

This is done by having a Haptic feedback object class inheriting from scriptable object. It has three public methods Light, medium and heavy vibration which are all wrapper methods around the Light Medium and Heavy Feedback methods provided by the CandyCoded library. We can add the CreateAssetMenu attribute to the class and can now create a HapticFeedbackObject in our project in unity. Now whenever a button is clicked for example we can drag and drop our HapticFeedbackObject into the unity event and select the appropriate feedback method. Alternatively we can hold a reference to this object and call its method through code like what is done for the haptic feedback on enemy kill.

As well as leveraging unique mobile features the game also takes advantage of various frameworks.

Those being the google play games services, cinemachine and unity cloud.

Google play games services are able to allow the user to authenticate themselves using their google play account. Save game allows the following data to be saved between play sessions. As the data is saved to the cloud it also allows the players data to persist if they was to uninstall then reinstall the game. Additionally, the google play games framework is also used to reward the player with achievements at various points of the game.

Awarding achievement is very easy through unitys Social API which provides a common point of interaction for multiple different social platforms. Social.ReportProgress is how you can award an achievement, you pass in an id which matches the id of one of the achievements set up in the play games console. A progress and a callback for when the process has completed.

Authentication is done through the SignInManager class on awake the PlayGamesPlatform is activated. Then on start we can attempt 0 click authentication. If the sign in is successful then we can invoke a unity event to let other objects know the sign in was successful. If it was not successful then we can prompt the user to sign in manually.

The next system uses the value reference class I talked about in this video and allows for a value of type color, float, int or bool to be stored in the assets of the project. This allow any game object to reference the same value and is very useful for configurable settings for example. The system is designed to be easly expamndable. And you can add new types for example a vector value reference.

As for the save game there are a few classes responsible for saving the game both locally and to the cloud. The brains of the operation is the BuzzardGameData class. This is a class following the singleton design pattern which is responsible for loading and saving all persistent data. It does this by reading and writing to 7 save files. There are files storing the current configuration of each of the parts of the ship, a file storing a dictionary which tracks the quantity of each item not equipped on the ship. And finally a save file which is a conglomeration of all of the different save files. Additionally containing player data such as the players money, games played and number of kills but also containing data configured in the settings menu such as if gyro is enabled or the color of enemy outlines. Finally the file stores the custom user sound to be played when the player dies in the main game scene. On start the buzzard game data instance sets up its dependencies this means loading all of the relevant save files and also loading all of the float, int, bool and color references that are used to store values any object in the game can access.

Next if there is a save game file its contents is passed to the OnSaveGameRead function

This function is used both to handle reading the local save file and reading the cloud saved file retrieved by the saveGameInteractor. If the data is empty then an initial save file is created by collapsing the current config Files and data to a string and saving this string to the cloud via the save game interactor then finally recalling the method but with no empty data this time. IF the data isn’t empty then we can save the data locally by first parsing it to a SaveGameData object and then passing it to the SaveConfigFilesLocal method. Which saves all of the composite save files and updates the different value references. It also saves the user sound if one exists. Finally the SaveGame file is written. Each time the scene changes all the save game files are saved using their current state and the game is saved to the cloud. The cloud save data is read once the main menu manager is notified that the user is authenticated. The buzzard game data then asks the G\_SaveGameInteractor to load and read the save game.

To demonstrate the save game data working I can go into the save game file and change some accessibility settings manually. Now when I load up the game you can see that the menu has reacted to what is in the save file. If we change the settings in the menu and then change scene you can see that the save file is updated.

When we load up the audio scene we can record some custom audio then change scene. In our explorer you will see our wav file, also in our save file you can see our wav file. If we delete the wav file from our explorer and start the game. The file is retrieved from the save file and resaved to the explorer.

Cinemachine is used instead of standard unity cameras in the main menu, upgrade scene and the main game scene. It is used to add camera shake into the main game when boosting as well as increasing the fov when boosting, additionally is used to add some dynamic movement to the camera in first person when piloting the ship. It is used to provide a third person mode to the main game where you can see the outside of the player ship flying. In the upgrade scene cinemachine is used to dynamically animate the camera when selecting a part of the ship to upgrade. The camera zooms into the focused area of the ship when selected.

Finally unity cloud services are used for cloud content delivery as well as unity advertisement. Banner ads are displayed when the loading screen is active and inside of the audio customisation scene. Additionally rewarded advertisements are used to give the user the choice to earn some additional credits they can use for upgrading their ship. When building I noticed that the applications file size is quite large. According the following google published report there seems to be a negative correlation between app APK size and install conversion rate for apps with sizes below 100mb. The research also finds that a significant decline in conversion rate is due to the install not completing. This is for a variety of reasons but an app with a size of 10 has a download completion rate about thirty percent higher than an apk of 100mb.

This can be from people having to pay for the data download, cancelling the download due to long download times or internet connectivity issues. Over 50% of Indian and Indonesia android smartphone users do not have access to wifi so likely ave to pay for any data they do use causing games with a large size less popular. And with android holding a 93.47% market share of the Indonesian and a 95.49% market share of the Indian operating system market reducing the download size of the game is very important.

Therefore unity addressibles and the unity cloud cdn is used to reduce the initial download size of the game. Additional content can be downloaded after the app has been installed from the play store. This reduces the size of the initial install something previously identified as important but also means certain elements of the game for example the player, enemies and scene can be changed without the user needing to update their game from the google play store. Instead the new content is uploaded to a bucket in unity cloud and is then downloaded when needed to the users device. This gives flexibility to me the developer and could facilitate seasonal changes to the games cosmetics for Christmas or haloween for example. As the addressable content is cached locally if the user doesn’t have internet access the old assets are used until internet access is back again and then the new content is updated automatically when the game starts.

https://medium.com/googleplaydev/shrinking-apks-growing-installs-5d3fcba23ce2

Github was used to version control the project. A kanban github project board in conjunction with github issues was used to organise tasks to be completed on the project. Github is also used to store documentation for the game in the repositories readme.

As for the planning of the game I use illustration to plan the flow of the game alongside reviewing other games to inform the design.

design mockups were also illustrated for the ui in the game. To design the ui of the game I also took inspiration from other games.

The game takes accessibility into account allowing the player to customize various parts of the game. These settings are accessible from the accessibility settings menu. The player can customise if the gyro is inverted, if the pitch input of the controls is inverted, the outline color and thickness of enemies and finally the color sceheme of the ui. Changing these colors can lead to some pretty cool new looking ui’s.

Thank you for watching here is some gameplay of buzzard.