Khaos Kitchen

A VPM Studios Production

Harvey Robinson (Project Manager), Finn Hobson (Lead Developer),
Aneesh Anand, Harry Goode, Alex Stedman, Felix Williams

May 2019

University of BRISTOL

KHAOS KITCHEN!

VPM Studios

Contents

1 Signed Assessment Page

Harvey Robinson

Project Manager
Contribution Weight: 1.07

Signed: _____

Finn Hobson

Lead Developer
Contribution Weight: 1.13

Signed: _____

Aneesh Anand

Contribution Weight: 0.83

Signed: _____

Harry Goode

Contribution Weight: 1.07

Signed: _____

Alex Stedman

Contribution Weight: 0.83

Signed: _____

Felix Williams

Contribution Weight: 1.07

Signed: _____

2 Top Contributions

The unique implementation of fast NFC into a video game.

Fun and fast paced multiplayer gameplay.

Scalable, robust low-latency networking and synchronised group gameplay activities.

High quality and variety of graphical elements and animations.

Real-time colour recognition.

Music, sound effects and real time feedback.

Effective implementation of agile Scrum and Kanban hybrid implementation throughout the project to work as an effective team.

Intuitive UI balancing clarity and engagement.

Use of phone accelerometer and microphone.

An immersive Games Day kitchen setup.

3 Abstract

Khaos Kitchen is a fast paced, cooperative local multiplayer mobile game, in which users become chefs in a crazy and nonsensical kitchen. Each kitchen can have 2 to 8 chefs on duty, but 4 to 6 chefs will create the optimal experience. A central screen shows the on-goings of the team's restaurant, where each player's chef avatar is working in the kitchen and customers are coming to dine.

Every round, chefs are given four action buttons that allow them to complete a specific set of actions. Chefs are also given instructions such as "Chop the Lettuce" or "Flambé the Strawberry" which have to be executed before the timer runs out. These orders may have to be performed by the user themselves or by any of the other chefs. This is where the hook of Khaos Kitchen comes in - a successful kitchen must have exceptional communication amongst the chefs on duty.

Each round of Khaos Kitchen requires the team of chefs to complete a certain number of tasks to keep their restaurant alive and progress to the next round. As the players advance, they will have to perform more tasks and the time allowed to complete each instruction decreases. Successfully completing tasks will keep customers happy and increase the rating of the restaurant. The players will be able to see their restaurant grow in popularity on the central screen. However, if the chefs make too many mistakes or fail to accomplish their instructions, the restaurant will be shut down and "Game Over" will occur. Their kitchen will be damaged and customers will leave the restaurant. Just like in a real restaurant, speed and accuracy are the keys to success.

The most unique aspect of Khaos Kitchen is that the game makes use of the full range of modern smartphone hardware features. The players are instructed to complete a variety of tasks using the accelerometer, microphone, camera and NFC scanner throughout the rounds. Figure 1 shows several examples of these alternative instructions.

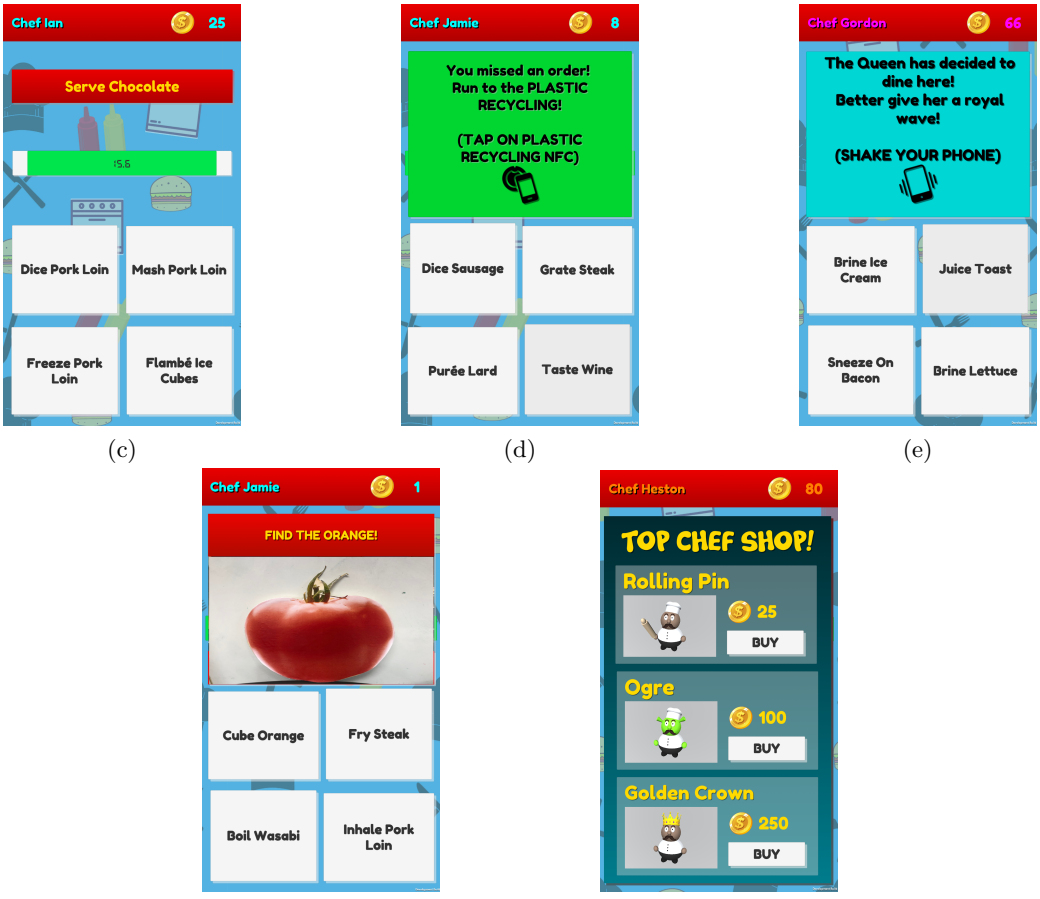

Figure 1: Screenshots of the Khaos Kitchen mobile user interface.

As well as completing actions on their phones and managing their virtual kitchen, players will have to run around the physical kitchen set-up that has been created. Throughout the game, the players will

have to interact with NFC beacons scattered around the room.

Although Khaos Kitchen requires players to work as a team, chefs will be rewarded with tips for their individual excellence. Each round, the most successful player will win the prestigious accolade of "Top Chef". This player will get the opportunity to spend their hard-earned tips on cosmetic upgrades for their chef avatar on the central screen.

Throughout the game, group activities occur which require coordinated actions from all of the players simultaneously. These instructions will appear on the central screen and the game cannot progress until the entire team has succeeded in completing the task.

Constantly switching from shouting to listening, virtual to real-world actions and cooperative to competitive interactions creates confusion and excitement amongst the players. The fast pace and growing pressure of each round builds the energy and khaos throughout the game. The combination of these features makes Khaos Kitchen a really enjoyable and challenging gaming experience.

Figure 2: Screenshots of the Khaos Kitchen host server interface

4 Team Process and Project Planning

4.1 Overview

The project structure followed an agile hybrid of scrum and kanban methodology. The project timeline is discussed in the following:

4.2 Initiate

4.2.1 Create Project Vision

From the two game pitches that were proposed, it was decided by the panel that Virgin Pornstar Martini Studios would produce the multiplayer Android mobile game, Khaos Kitchen. Although this was the second choice, the team was excited to begin planning and development. A group brainstorm meeting was held, resulting in a number of ideas for what Khaos Kitchen would be like.

A fast paced, khaotic theme would be the most applicable to the project. It allowed for multiple phone features to be deployed, as well as ensuring the game was enjoyable to play, through its fast pace and khaotic humour. Technology involved was confirmed to be up to 6 mobile devices, utilising all phone capabilities accompanied by NFC tags to bring the game into the real world. The Android application market is very saturated, however due to the awkwardness of setting up NFC beacons, very few games involve them. This is where the game proposal was unique, as the only team to propose this originally.

4.2.2 Develop Themes

The first considerable section of work was assigned to research and learning. Working through online tutorials allowed members to understand how unity set up and managed a game. A dummy game was created to hone members' skills.

The first core section of the multiplayer game was setting up the network. This was broken up into the generation of host setup, client setup, lobby UI, experimenting with `SyncVar`s and `Client RPC` calls.

Another core section was to make a basic framework of the game. This theme ran concurrently with the lobby, as neither required the other to proceed. A single player prototype was created as an extension to a previously undertaken tic-tac-toe tutorial. This used a `GameController` script to produce instructions and handle logic, with listeners for the microphone, accelerometer and NFC scanner.

Having completed these two themes, integrating them produced a very simple two player networked game. The next step was to add features to make the game more playable, resulting in the Minimal Viable Product that was displayed in January.

The next two themes split into server side and client side. The server side theme involved modelling a kitchen in Maya along with animations. The other theme was a complete overhaul of the system architecture. This was required to allow for further functionality to be integrated into the game.

Following this, the final theme was adding complexity to the gameplay.

4.3 Plan and estimate

4.3.1 Create User Stories and Tasks

At the beginning of each sprint, the team manager came up with a list of priority tasks for the week and then a meeting was held create and discuss a plan for the following sprint. The main objectives of each sprint were melded into stories, then dissected into smaller, more manageable tasks. As part of the discussion, these were assigned a priority before being assigned to either to an individual or pair. In most sprints, the assignment of tasks was achieved by people electing which story they would like to work on, with the last person for that week getting the last available task. As team members became subject matter experts within the group, they often continued on work they had experience with.

4.3.2 Approve, Estimate, and Commit User Stories and Tasks

Tasks were attributed and estimated number of hours for completion. Before meeting with Anthony, the groups mentor from CACI, stories were assigned with very few tasks, and often far too ambitious for a single sprint. Anthony helped to understand how to break down each story into manageable tasks by assigning tasks with estimated number of hours. Throughout the year, as experience grew, these estimates became increasingly accurate, to the point where they eventually became more realistic.

Gitkraken's globoard was used as the scrum board. This was organised into 4 sections: Todo, developing, testing and completed.

4.3.3 Create Sprint Backlog

As the tasks and sprints refined in accuracy, there was opportunity for those which completed stories under the estimated hours for the sprint to pick up extra tasks. These tasks were taken from an additional stack of lower priority tasks was created during the weekly meeting.

4.3.4 GitFlow & Work Flow

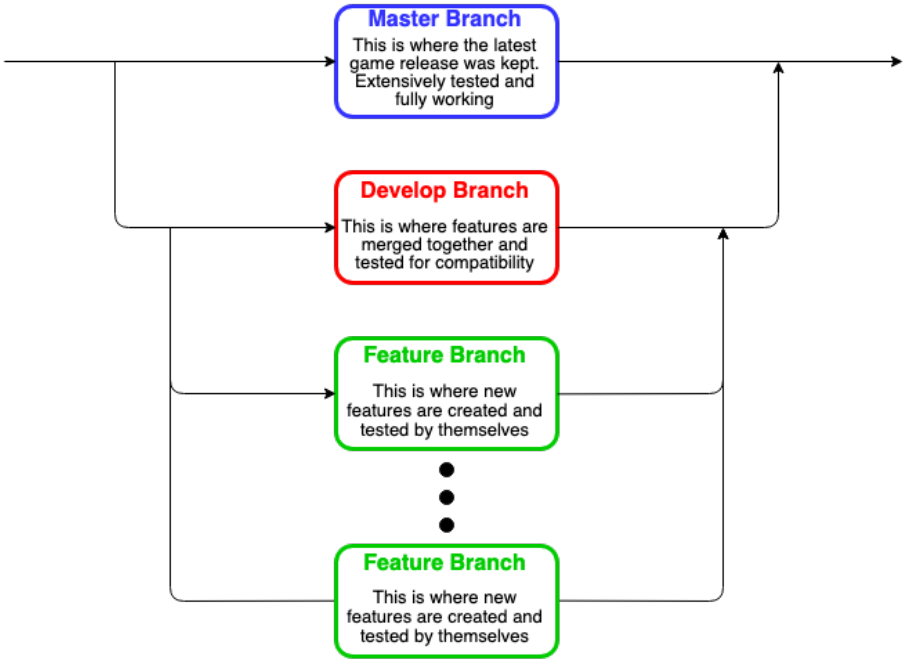

Figure 3: GitFlow and Branches

A GitFlow was devised by the group and is represented in Figure 3. It involved a master branch with releasable builds, a develop branch where tested features are merged to test for inter-feature compatibility and feature branches are used to develop and test new game features by themselves.

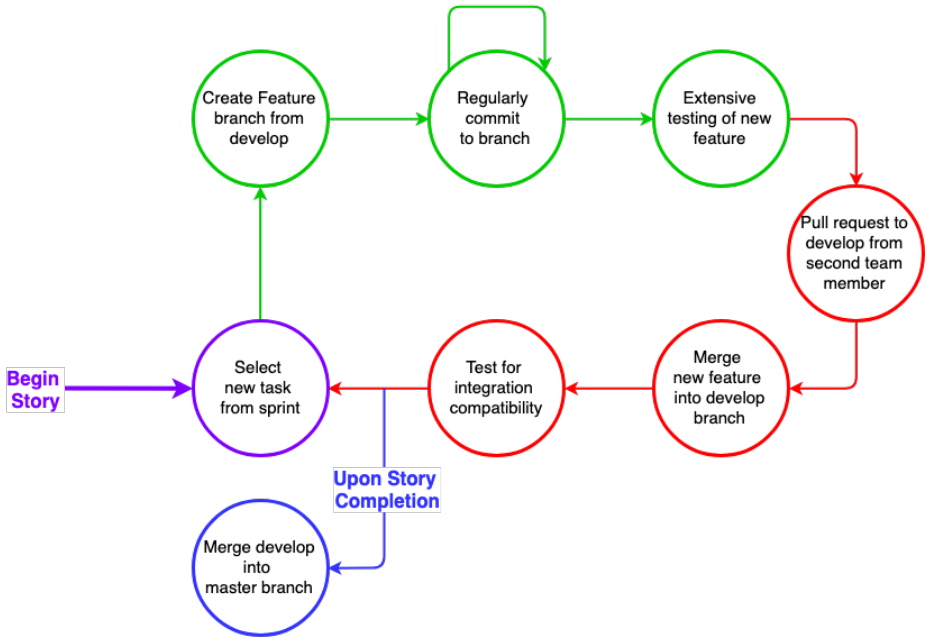

Figure 4: Work Flow

From the GitFlow defined by the group, it was possible to create a work flow for members to follow for stories and features. This meant issues from using git was minimal and allowed for a large amount of separate features to be developed concurrently.

4.4 Implementation

4.4.1 Code review and approval

Code reviews were completed as part of the branch merge process. On a pull request, another team member would have to review and approve code changes as well as check the merge is working. Once approved this would be merged into the develop branch. This made multiple members accountable for changes but more importantly reduced accidents. This ideally was done when team members were apart, however due to the nature of the game and the reduced number of testing devices, this had to be done in person.

4.4.2 Issue handling process

When problems arose, or if a task was discovered to be too lengthy or complex, the issue was raised with the teams group communication mediums. These were then discussed and handled mid-sprint. If the issues were deemed too difficult for the remaining time in the sprint, other tasks were assigned and the issue was tackled in smaller sections in the following sprint. If the issues were deemed unrelated to the current sprint, they were added to the backlog of low priority tasks. This stack of tasks was consulted at the planning stage of each sprint.

4.5 Review and Retrospective Sprint

At the end of each sprint, a review session was held. Each member discussed the work that they completed in their sprint. A key part of the discussion revolved around what succeeded and why, as well as what did not succeed and why. Due to the estimated nature of a sprint, part completion was not uncommon. However, the importance of learning from these failures was the focus. These sessions helped refine estimates for the planning stage of the following sprint.

4.6 Retrospective Positives

4.6.1 Weekly catch up / Accountability

Catch ups were regular, members were held accountable to the hours agreed. Each sprint member had learnt new techniques on how to solve specific problems, and the retrospectives allowed for this knowledge to be shared.

4.6.2 Paired programming

To the best of the groups ability, sprint stories were assigned to pairs. This aided in tackling the sprint challenges, as well as making the group more versatile to members other commitments while at University. This was used for the majority of tasks that required discussion however some tasks would have been a waste of manpower to have two people working on so only one member was assigned. The benefits of pair programming were very evident as it decreased time to solve individual problems as it provides effectively a continuous code review. Two problems presented themselves from pair programming, sometimes one member would take over on tasks and leave the other member not following what is happening if they're not communicating well as they coded. For pairs who struggled with this, extreme programming tactics were exercised such as hot-keyboarding where the members would switch who is in the driving seat every 20 mins strictly, even if mid line. This ensure both members fully understood everything that was coded and prevented members turning off.

At the beginning of the project, each week members were paired with different partners, ensuring all learnt basic skills for the project, as well as rejuvenating group energy.

4.7 Acquired understanding

4.7.1 Remote working

During university holidays, the groups team process had to be adjusted due to the team being unable to be together in the same physical location. Due to the limited number of phones available (4) to the team, only 2 people could reliably test deployment on phones, as it requires a minimum of 2 phones to be able to play.

4.7.2 Unity

Unity prefabs are binary files and therefore not able to be merged through git. This proved to be an issue when two stories required altering of the same prefab or UI element. Issues arose at the beginning before this was realised, with entire stories being lost. This resulted in careful consideration of stories so as to ensure no two stories required work on the same prefab.

4.7.3 Pre-learning

At the start of the project, no time was initially allocated to learning the required skills to use the unity engine. If the project were to be attempted again, more time would be allocated to learning and honing the required skills for the project before development begun.

4.7.4 Spaceteam

The group were lucky to get into contact and liaise with the creator of the popular app Spaceteam. He gave advice as to the best ways to manage local multiplayer games. It was concluded that Bluetooth was not robust enough for the kind of game envisioned hence processes to create Local Area Network went ahead.

4.7.5 Task assignment

During the weekly meetings plans were made for the upcoming week and stories were assigned to team members to be completed in their next sprint. For the majority of development, task were assigned to pairs as it was thought that the highest level of productivity was achieved when working in pairs.

4.8 Test driven development

4.8.1 User acceptance testing

Mobile testing proved to be an arduous process, with the main bulk of this testing consisting of user acceptance testing. Build times grew as the graphical element of the game became more extensive.

4.8.2 Unit Testing

The majority of the testing process for Khaos Kitchen was conducted throughout production through a combination of user acceptance testing and writing to the console to ensure that the required outcome was being achieved. Especially for the case of the multiplayer networking it seemed more logical to have a console message for a successful connection.

However, a collection of unit tests were written to ensure that any new code did not affect the performance of the gameplay mechanics. This ensure the codebased remained stable, automatic unit tests were executed before building the game. In particular this included how the score was being increased for correct instruction completion and that crucial gameplay variables such as *Round Timers* and *Number of Players* were being initialised correctly.

4.8.3 Testing days and reviews

Testing day was a worry for the group, as this was the first public beta tests. The focus of Testing Day was to work on the gameplay, with an objectively ugly UI, the aim was to see if the gameplay loop was enjoyable. The game was enjoyed by players of all ages, “Pretty much the perfect game!” quipped one child, “The only game without any bugs” another retorted. Although the feedback was overwhelmingly positive, two issues in how the game was being played were noticed. Players were largely stationary and huddled together for most of the game, and they rarely looked at the projection of the kitchen. This was not how the game was designed to be played. To amend this the probability of a phone interaction action (shake, shout, scan or NFC) was greatly increased. This meant players were now running around the arena more and provided a higher sense of urgency. Group instructions were also added at this point. They required the players to look at the kitchen projection in order to receive their instruction and it displayed a leader board of the order of player completion. This ensured players frequently had a chance to observe their kitchen and provided a unification of the physical set up, mobile phones and the projection of the server screen.

4.9 Team equity and accountability process

To keep the teams equity fair and to resolve potential issues during development, anonymous equity share votes were held via Google forms three times during the development year, after MVP demo, Easter review and final submission. These would be an equity vote on the prior term with the final vote also having overall equity vote to ensure some fairness and comparison. After each of these votes, the given equities were revealed to the individual and they were able to have a discussion in person as to the result. This was an overall discussion where good and bad points were fed back. Before Easter and Final review, a team discussion was held as to the submitted equities so all team members could discuss the equities to decide they were fair.

5 Individual Contributions

5.1 Harvey Robinson (Project Manager)

Core game and logic (130 hours)

- Researched and Implemented networking via conversion of solo version to multiplayer version of game using HLAPI.
- Created original button distribution, game logic and startup game logic.
- Created logic for random distribution of buttons to individual players and original ordered recipe logic.
- Implemented both group NFC race and group shake instructions logic and display

NFC script, implementation, re-implementation and create beacons (80 hours)

- Researched how to implement and use a NFC technology, created a separate proof of concept NFC app in unity.
- Created generic NFC listener script.
- Spent time fixing the double NFC scan bug. Due to little success, implemented no double nfc scan logic via NFC bin and serve logic, 2 bin tags for missed instructions and 2 service tag for streaks of hit instructions.
- Re-implemented NFC for more generic amount of NFC stations and grouping into sections of NFC. Server would allocate NFC section but user would allocate precise NFC beacon for increased movement.
- Increasing NFC sensitivity via physical and technical solutions including different NFC tag sizes, NFC lattice research and creating an optimal NFC beacon in the hackspace.

Project manager roles (15 hours)

- Generic admin,equity voting and team feedback.
- Creating weekly task of priority tasks. As Scrum Master, creating and policing scrum board.
- Implementing good practices and methodologies learnt via two prior industry software internships at HM Government and J.P.Morgan such as using a scrum board, scrum/kanban hybrid methodologies and an attempt at extreme programming ideas.
- Inputting git restrictions of pull requests requiring a second approval.

Team process (45 hours)

- Discussion of weekly tasks, group direction and meetings with Anthony/Tilo.
- Debugging other members laptop to phone glitches as for much of development only four phones were available, two of which normally in my possession due to my tasks.

Low level networking (25 hours)

- Researched and created example low level TCP networking unity project with multiple users using LLAPI implemented as a star topology.
- Started to implement into game but upon realising this could not be used half and half with the existing networking code as they interfered. This was this was put to a vote as to whether to implement a full switch but was voted against due to time constraints and HLAPI being adequate for the game needs.

Game play testing and fixing bugs (50 hours)

- Generic playing of setting up the game in physical locations, testing the full game and balancing.
- Fixing game breaking player count and networking bugs acquired from reduced testing from either only laptop builds or not playing long enough.

Misc other (56 hours)

- Basic Unity familiarisation by developing simple single player mobile game (20)
- Wrote entire team process section and co-wrote relevant sections of technical followed by team review (25).
- Minor Networking and code optimisation but it was decided to reduce the amount of data synced across between the phone clients and laptop server as well as excess/ inefficient code. (5)
- Implemented constantly updating server logic between players from static lobby page to synchronised consistently updating lobby page.(5)
- Extensive and lengthy discussion about the team name occurred.(1)

5.2 Finn Hobson (Lead Developer)

Multiplayer Networking (75 hours):

- Along with other members of the team, I researched ways that we could implement our multiplayer networking system. We decided that Unity High Level API would be the most suitable and effective system to connect our player clients and allow them to communicate with a dedicated server throughout the game.
- I was involved in implementing the lobby of Khaos Kitchen. I wrote scripts that connect the players to the dedicated server and designed elements of the user interface.

Gameplay Prototyping (50 hours):

- I worked on the initial design and implementation of our software architecture, our core gameplay features and our gameplay flow. This involved using ClientRPCs to distribute instructions and actions from the server to the player clients; using Commands to let the player clients communicate with the server when an action is performed and implementing the basic gameplay logic.

Microphone and Camera Features (75 hours):

- I worked on integrating the use of the phone microphones into the Khaos Kitchen gameplay. This was challenging due to hardware issues, but myself and other members of the team managed to successfully add this feature to the game.
- I wrote scripts that allow seamless and instant access to the phone cameras during the Khaos Kitchen game, and scripts that perform real-time colour recognition on the camera input. I integrated this feature into the gameplay flow.

Graphics and Animations (150 hours):

- I worked on designing the Khaos Kitchen chef and customer characters and modelling these prefabs on Maya.
- I wrote scripts to handle the spawning, movement and animation of the kitchen and character prefabs.
- I used Unity's particle effect system to create fires in the kitchen when the team of players make mistakes during the game.
- I worked on implementing the 'Top Chef Shop' feature which allows users to customise their chef avatars throughout the game.
- I designed the user interface on the central screen, displaying scores, timers and other key pieces of information to the team of players during the game.

Team Process (50 hours):

- As a team we had regular meetings to discuss various aspects of the Khaos Kitchen game and our software development process. As lead developer, I often lead the discussions regarding software and development decisions.
- I was the lead correspondent with our composer, Constantine, and regularly communicated with him about the team's ideas for how his music could enhance our game.
- I carried out extensive user acceptance testing of our game along with the rest of the team. This allowed us to test our features for bugs and determine the best possible gameplay balance and difficulty level.

5.3 Aneesh Anand

Multiplayer Networking (100 hours)

- I researched with another team member into the Unity High Level API networking system and the various different options that were available for Khaos Kitchen's networking system.
- We as a pair decided upon using a dedicated server as the host screen and the phones entering as clients.
- Collaboratively we wrote the code for the lobby so that the dedicated server could "Make a kitchen" and then clients could "Find a kitchen" by entering the required IP address in a small text box.

System Architecture Restructure (50 hours)

- After the January review it was decided amongst the group to adjust the system architecture to have a separate `InstructionController` dedicated to the distribution of instructions amongst players.
- I contributed with two other team members with the planning and then implementation of this new system within the game. This in turn greatly simplified the architecture of Khaos Kitchen and not one class was ordering every bit of computation.

Testing and Debugging (100 hours)

- I wrote all of the unit testing for Khaos Kitchen which predominantly tests that in-game gameplay variables are being changed in the correct way such as score being incremented and round timers decreasing correctly.
- This meant that when new code was developed, we could always run the automatic unit tests very quickly to ensure that the core gameplay features were still in tact.
- Once new code was developed I also wrote more tests to ensure everything was being completed as expected.

Top Chef Feature (25 hours)

- I wrote the code for the first implementation of the "Top Chef" feature within Khaos Kitchen which was later tweaked when the "Shop" was implemented.

Team Process (101 hours)

- Brainstorming sessions were held early on where we set out our plans for the project, including future game features.
- Meetings were held every Monday where we set our tasks for the upcoming sprints
- Any separate features that were ready and fully tested were then merged into our develop branch.
- I contributed large amounts towards the game's user acceptance testing and debugging. Significant periods of time were also spent completing our feature branch merges, which often led to lengthy sessions of debugging and refactoring code.
- Team name discussion: (1 hours)

Documentation (25 hours)

- I spent time into writing up large amounts of the report, specifically the technical content section, as well as helping prepare for the presentation.

5.4 Harry Goode

Core game logic: *(150 hours)*

- Created core game logic of clients and server communication.
- Created the logic of instruction handling, player actions and rounds using SyncVars and ClientRpcs.
- Rounds logic to halt the game actions at the end of each round, collect round data and then restart next round.
- Created NFC station logic to ensure robustness and uniqueness.
- Generated game logic to initiate and process the group activities within gameplay.

Created the Instruction Controller script: *(100 hours)*

- Handled the distribution of actions and instructions at the beginning of the game and at the start of each respective round.
- Handling logic for processing player actions, and proceeding them
- Handling multiple player logic, distributing new instructions when actions were completed, as well as allocating correct scores.

Networking: *(30 hours)*

- Integrated lobby into the game.
- Created logic such as lobby hooks to transfer data entered in the lobby by users to players.

Game feedback and music: *(20 hours)*

- Sound effects, colour indicators and haptic feedback were implemented to add to gameplay. These were set on triggers to ensure real time feeback.
- The sound effects and music were controlled from a music player script I wrote.

Other team issues: *(100 hours)*

- Time spent doing team merges and further testing took considerable amount of time. The reasons for this are talked about in other areas of the report.
- Weekly meetings and discussions ended up accumulating hours, but this was important to ensure all members were clear on the project vision and sprints.
- User testing contributed a huge part of the time of this project. Time spent finding bugs and fixing were a huge part of the game.

Miscellaneous: *(1 hour)*

- Team name decision was a serious topic. It set the tone for the project.

5.5 Alex Stedman

Working with Virgin Pornstar Martini Studios over the past year has been a fun and rewarding yet challenging and difficult experience. It has been a long process to finally achieve the desired result we as a team envisioned, but I believe that working game we have now is a real experience that has come from copious hours of game development and bug fixing.

We spent a good 60 minutes discussing our team name. After this the first task I had as part of the team was attempting to create a click to move programme that we would be able to implement into our game. At first we envisioned chefs on each phone that one would be able to interact with by clicking on the screen. However, a short while into the first term we realised that the phones simply were not powerful enough to handle such graphics hence the idea was put on ice. Unfortunately around (30 hours) went into this work that was never used but positively it had given me a chance to really work my way around Unity. The team instead decided to concentrate on a central graphics screen with animation, and a fun and engaging UI on the phones. This became my main concentration for a good part of second term.

I created a plan to work out the best way to create fun and interesting graphics. I decided to first polish the central screens graphics and animations then move onto the phones UI. I used Maya to create around 20 models that would be used in the game from the chef and customer prefabs down to the pots and pans to add to the look and feel. The task was harder than I had first thought, as I really wanted to make the graphics have its own sense of togetherness and identity. As well as this, although Maya and Unity work very well in tandem, sometimes it was tricky to get the imports working correctly with some parts of Maya not being supported. Over time though the process became more streamlined and I was able to create the models in around (40 hours) I then had to design the kitchen itself and put everything into place, applying prefabs and the like, which wasn't a long process and took around (10 hours).

It was then a task of actually linking all the objects and prefabs I had created into the game. Scripts such as the `Animation Controller`, `Game Controller`, `Customer Controller`, `Player` had to be modified to allow for the customers and chefs to spawn and for their logic and movement in game to work correctly. I worked alongside each other and this was a long process from beginning to end. The initial implementation took around (70 hours). However we then realised that the prefabs looked strange in their current standard gliding and all looked the same. This was when we first started looking at animation. We wanted all the chefs and customers to have their own unique movements such that they came alive. I created idle, bouncing and celebrating animations and used the animator and the `Chef Animation Controller` to implement them using logic in game. This took another (40 hours).

After we thought the main screen was done and just required some polishing (another 20 hours) it was then time to focus on the phones UIs. I made myself comfortable with GIMP and went about designing the background and general layout of the phones. I found some pre-designed buttons and was able to import them into our game. Re-designing the whole lobby and player scenes was a long and tedious task but one that in the end gave a result where we could see real changes as a team. Only small amounts of back end implementation were required at this stage, such as small animations to make the buttons wiggle and a 15 line script to spin our logo. This whole process took another (30 hours).

Once these tasks were completed it was mainly a case of refining and debugging. I helped fix and sort out player movement bugs such as customers moving through tables and facing the same way. On top of this we fixed general bugs as a team and as partners throughout the majority of the end of second term and end term. Small things like the timer freezing when a group activity appeared, to the camera not picking the correct colours, all these tasks I helped and aided looking at the code and physically testing to see if the problem had been fixed. This all obviously takes quite a long time and added another (150 hours) of work onto the table.

Right at the end of the process I helped with the physical creation and design of our NFC beacons and images to make the game as fun and immersive as possible. We went to the hackspace and spent (2 hours) or so creating what we needed.

5.6 Felix Williams

Game Implementation: *(200 hours)*

- Created the initial single player prototype. A simple random instruction generator where the user responds through fixed button or the use of phone senors. It contained the listener scripts for the microphone, accelerometer and NFC scanner; with accompanying game logic to check and handle these inputs. These are still used in the final build and the single player prototype contains the gameplay flow that all later releases followed.
- Created the logic to use SyncVars and RPCs for communication, accompanied with logic for round progression, failure and success including timers for instructions and rounds on the first networked prototype that was then replicated by other team memebers to produce the January prototypee.
- Devised the system architectures of players with listeners interacting with a game controller and instruction controller and assisted in the implementation of converting the January prototype into the architecture.
- Devised and created the difficulty algorithms that responds to both round number and player count, that set instruction time and instruction number each round. Along with a static script to store settings that can be changed from the lobby at runtime on the host computer. This allowed altering of the difficulty algorithm, player count and various other hard coded values. This drastically sped up the testing process by removing the need to rebuild and redeploy the application to devices when changing settings.

User Interface and Experience: *(50 hours)*

- Designed and created the user interface for both the home screen and during gameplay. This included designing buttons and panels to be clear and informative, yet also aesthetically pleasing. I edited the graphics of several buttons, previously in the game that were imported assets, in order to fit with the colour design of the game. In this service I also created idle animations for most UI elements, including the buttons, home screen quotes and logo, and created a panel behaviour script to simulate an infinite scroll background, after also changing the colours of the designed image. I ensured these animations were based off time and not frames to ensure smoothness and prevent differences between computer and laptop builds.
- Designed and created the random instruction generation system that provides the game with a level of humour. To assist this I created the instructions used for “Shake”, “Shout” and “Scan NFC” instructions and home screen quotes with a similar emphasis on ridiculous humour. This resulted in a total of over 1500 possible instruction actions, greatly increasing the replay value of the game.

Team Process and Debugging: *(151 hours)*

- A large amount of time was also spent on extensive and lengthy merges, including manual merges of prefabs where work had to be reproduced due to merge issues from binary files. As well as reviewing other team members code and assisting in merging their features into the working develop.
- A significant portion of time was spent debugging as crashes and freezes could be down to any number of issues and getting debug reports from the devices was troubling. There where also several issues from differences in computer and mobile builds as well as networking bugs that required extensive debugging and testing.
- Weekly meetings cumulatively took a large number of hours as there would often be lengthy discussions about the vision of the game, or merge issues that needed work. As well as discussions about the weeks tasks and who best to work on which sections of the game.
- Extensive and lengthy discussion about the team name occurred.

6 Software

6.1 Building

Khaos Kitchen was developed using Unity. This game development platform was chosen due to Unity's abstraction of low level detail and its excellent cross-platform development features.

Unity provides the features for building the game for both PC and Android. The game needs a PC to host, but the players can be either PC or Android. The host must click "Make a Kitchen", whereas the players must click "Join a kitchen". Pre-built versions of PC and Android have been provided, as errors and bugs arise from different builds interacting.

Before running a new build, whether it be on Android or PC, all of the unit tests that were written were always run to ensure that none of the core functionality was significantly altered. This can be done by bringing up `Unity`'s test runner and running all of the automatic unit tests.

6.2 Deploying

Having built the application in Unity, simply transfer the file to the Android phone, select and press install to build the application on the Android device.

To start the game, choose one PC to be host and select "Make a Kitchen". Then select a minimum of two devices, PC or Android to act as players. Type the IP of the host machine into the input box at the bottom of the screen and then select "Join a kitchen".

This will move the players to the lobby, where they are required to enter their name into the input box. A house icon will appear next to the input box that each user needs to use. Once all users press the "JOIN" button, the game will start.

6.3 Adding functionality

Adding new features to the game involved creating new scripts or editing existing scripts. The new features purpose would determine which script they would be appended to. For example, any features relating to the generation or handling of instructions would be added to the `InstructionController` script, changes to the main gameplay logic could be made in the `GameController` script and features relating to the players or Android devices could be added to the `Player` script. Please refer to the System Architecture diagram later in the report for further detail on inter-script behaviour.

Another way to add features, particularly relating to UI, would involve changing the prefabs. The Unity editor provided a GUI with sufficient tools to do the job.

The final version of the game was developed in Unity version 2018.3.6f1. The code is written such that functions names should be sufficient to understand the function's purpose. This greatly removed unnecessary comments.

6.4 Version Control and Feature Merging

Git was used for version control. A sensible GitFlo was devised for group development. To avoid code merge issues, a master and develop branch were created. The master held releasable code, and the develop held the newest working version of the game. No member was allowed to work on the develop branch, rather features were branched off the develop and then merged back in. GitKraken was useful here, utilising the "Git Glo" tool produced a well organised repository, allowing for concurrent work.

It was imperative that every member of the group understood this process, as to minimise errors caused from erroneous git calls. The GitKraken Client was used to produce a graphical interpretation of the branches. This was a big benefit to inexperienced git users within the group.

However, git did not provide tools to merge the Unity files, such as `.prefab` file types, due to them being binary files. This caused merge issues, and the resultant solution was to ensure that, as much as could be helped, members editing the same binary files was minimised within a sprint.

Once a feature had been completed, it was tested by other members not involved within the sprint. Satisfying testing meant this feature was then merged into develop. Further testing followed, which checked the merge had not broken anything, and if so issues were resolved as a group.

7 Technical Content

7.1 Phone Capabilities

One of the unique technical features of Khaos Kitchen as a mobile game is the way it utilises all of the features of modern smartphones. This was very much the trend around 2010, however since then mobile games have seen a decline in the use of inbuilt sensors. The accelerometer sensor, microphone, NFC reader and camera of the provided Android phones have all been integrated into the Khaos Kitchen gameplay. Throughout the game, players will be instructed to perform various actions using these sensors. Listeners have been created for all of the features, which are called when a relevant instruction requires them to. These listeners take input through their specific hardware input device and process the input, then the game logic performs the corresponding response.

7.1.1 Phone Movement

The accelerometer sensor is utilised when players are given an instruction that requires them to shake their phone. The `ShakeListener` script constantly checks the acceleration vector of the phone's accelerometer. If the magnitude of this acceleration vector exceeds a threshold, a shake is registered.

The input data of the accelerometer was very noisy due to the high sensitivity of the hardware. This caused small amounts of movement to register as a successful shake. Therefore, a low pass filter was applied to the acceleration data to reduce this noise and increase the accuracy of the `ShakeListener`.

7.1.2 Microphone

The microphone comes into use when a player is instructed to shout into the phone. The `MicListener` script accesses the microphone of the Android device, records a short audio clip and checks if the maximum amplitude of the audio clip exceeds a threshold. If the threshold is in fact passed then the microphone listener will signal to the `GameController` that the shout instruction has been completed.

Initially, the `MicListener` was permanently active throughout a game of Khaos Kitchen. This was a problem because after a few minutes, the Android device would automatically disable the microphone due to inactivity, preventing the `MicListener` from working for the rest of the game. To fix this, the `MicListener` is now disabled by default and the `GameController` enables it on a player's phone when the player is given a shout instruction. If a shout is successfully registered, the `MicListener` is disabled again until the next time it is needed. The dynamic activation allows the `MicListener` to be used accurately throughout a game of Khaos Kitchen.

7.1.3 NFC

Throughout the game, players will receive frequent instructions to scan specific NFC tags around the physical setup, simulating collecting ingredients from around the kitchen or serving the food to customers. Correct NFC readings are detected by the server through the utilisation of the `NFCListener` script, which returns the stored data string of the scanned NFC. The script works by monitoring the `android.nfc` object which stores all data associated with NFC's. The byte payload of the last scanned NFC is converted to a string and accessed through a public `get` function.

7.1.4 Camera

At various points during the game, players will be instructed to find different food items around the physical setup. The camera will become active on their phone screen for them to point at the specified item. The `CameraController` script accesses the back camera of the Android device and displays the camera input on a panel on the player's screen. The team discussed performing various methods of image processing on this input, but in order to fit with the fast pace of the Khaos Kitchen gameplay, it was decided that the image processing had to be real-time. Therefore, the most suitable image processing method was colour recognition.

The `CameraController` converts the camera input into an array of pixel colour values. Initially, the average RGB value of the pixel array was calculated and compared to a selection of colours. If the

average RGB value matched any of these colours, a signal would be sent to the `GameController` that the colour was detected. However, when pointing the camera at an object, noise from the background and the lighting would distort the average colour value, meaning colours where not always accurately recognised.

Instead of this, each individual pixel is compared to the selection of colours and if a significant proportion of the pixels match a specific colour then a signal is sent to the `GameController` that the colour is detected. This method was much more accurate at recognising colours, even with noise from the background and the lighting. This new method was slightly slower than the original method, but through optimising the code it still runs in real-time without any delay.

An issue similar to the microphone timeout issue would happen with the `CameraController` where the camera would be automatically disabled by the Android device hardware after a period of time. Therefore, the `CameraController` is also dynamically activated on the players' devices throughout a Khaos Kitchen game.

7.1.5 Immediate Feedback

For all of the phone features that are being utilised, immediate feedback is retrieved by the players which can be haptic or in the form of audio. This in turn meant that the players will easily understand that the action being asked of them has been performed correctly. As an example, correctly reading an NFC tag as requested produces an audio ring to notify the user of a completed instruction. Conversely, an incorrect button press causes the button to turn red, the phone to vibrate and a negative-sounding audio queue to be played.

7.2 Multiplayer Networking

Due to the required communication and synchronisation of the Khaos Kitchen game, a solid networking solution was required. After researching different networking options, the team decided that Unity's High level API would be the most suitable option for Khaos Kitchen. It provides sufficient abstract functions to facilitate simple, but robust communication over a local area network.

7.2.1 Server-Host Model

The network is constructed with a dedicated server handling game logic, and clients handling user inputs. The network architecture for Khaos Kitchen was designed to keep the communication overhead minimal, ensuring the passing of information between relevant parties is as simple as possible. The gameplay required no direction data communication between players, as this is a feature to provide challenge to the users. From this it was decided to have the clients communicate data with the server only. All game logic is then processed on the server. This format of communication can be seen in the architecture diagram in Figure 5.

7.2.2 Synchronising data

Unity provides several abstract methods to communicate information between clients and server.

For simple data, `SyncVar` variables are utilised. Upon compilation, Unity generates functions that push the variable values from the server to the clients. This is used to communicate simple shared data, such as score.

Some `SyncVar` variables have hooks attributed to them. These functions are called when the variable values are altered. This ability is useful for triggering events, such as round winner announcements.

More complicated communication was processed with `ClientRpc` and `Command` functions. The `ClientRpc` function is called by the server, and using data from the server, runs the function on each client. The `Command` is a function called by the client, and run on the server using client data.

Utilising these various forms of communication amongst the relevant parties throughout the game meant that information could be passed around seamlessly with a very low latency.

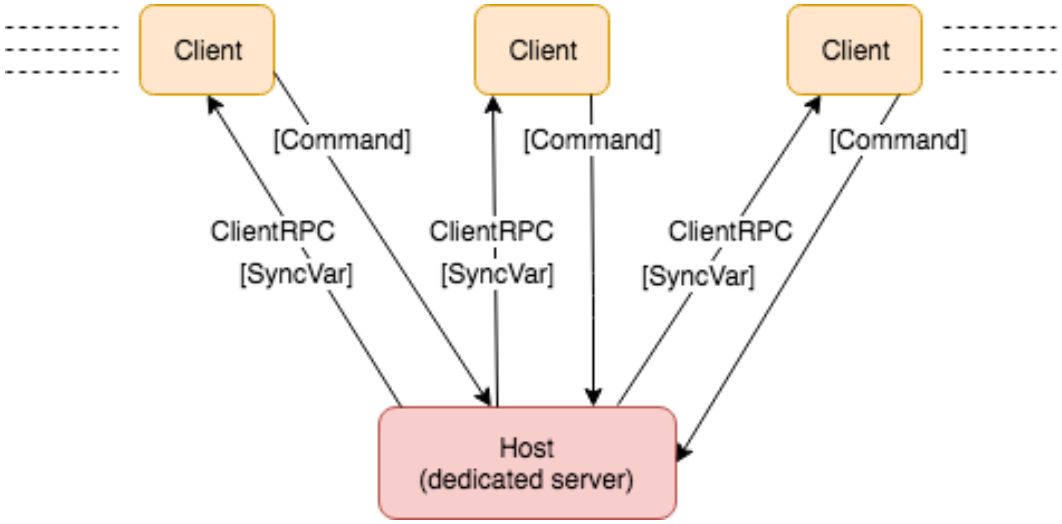

Figure 5: Network architecture diagram

7.2.3 Lobby

To start the game, a host must initiate the lobby by "Making a Kitchen", and users can enter the lobby as clients by "Finding a Kitchen". From the host screen, a settings page can be used to set the desired number of players for the game and to also adjust the difficulty through multiple variables that control the growth and behaviour of the difficulty algorithms.

This ability to change these variables was invaluable during the testing and fine tuning of the gameplay balance and difficulty. An optimal balance of starting difficulty was calculated in advance by the team.

On entrance of clients to the lobby, users can enter their name to be used within the game. The lobby also assigns each client an unique colour, to ease identification during the fast paced game. The colours are consistent for each player across all game items that relate to said player. This rapid visual association keeps the fast pace in various aspects of the game, such as the NFC race to tell the players as to which NFC station they need to run to.

7.2.4 In-play synchronisation

The minimalist nature of the communication between the clients and the server meant that the instructions were received by the clients in real time. Khaos Kitchen requires a minimum of two players, and although it has only been tested with a maximum of six due to a limited number of resources, theoretically the number of players is limited either by the power of the host computer or the bandwidth of the network. However it is thought that exceeding 8 players hampers gameplay due to the emphasis on communication between users, and 5 players is optimum for difficulty paired with enjoyment.

7.2.5 Group Activity

It was realised within the group that due to the low level of latency, accuracy to the nearest frame, it would be possible to include some group activities within the gameplay. From this a synchronised "shake" and "NFC race" were implemented to add to the theme of Khaos Kitchen being a team co-op game.

7.3 Graphics and Animations

The development of exciting and engaging visuals was highly important for enhancing the Khaos Kitchen gameplay experience. Graphics and animations have been used to give the players a clear indication of their success or failure throughout the game, as well as increasing the fun and khaotic atmosphere.

Having taken graphical inspiration from a plethora of games such Cooking Mama, Animal Crossing and Overcooked, the vision was to create visuals that were simple, funny and adaptable. The team decided on an art style that was cartoony, geometric and exaggerated.

7.3.1 Process

Autodesk Maya was used to create the 3D models that are used in Khaos Kitchen. This was a useful tool due to its support for importing models into Unity.

There were two areas in which Maya was used to create the game's visuals. Firstly, the team wanted to create a 3D restaurant environment in which the players would see the virtual representation of their game progress. The team decided to create diner style restaurant in which chefs would run around an ever busier kitchen whilst more and more customers entered the dining section. The kitchen outline, walls, interior and exterior were first modelled as a starting point to this scene.

7.3.2 Kitchen Design

The team then began to design the objects and surrounding scenery that would create the desired look and feel. The majority of the 3D objects in game were designed and modelled from scratch in Maya. The building model was filled with objects that make it look like a restaurant and kitchen. Figure 6 shows an image of the 3D restaurant environment that is shown on the central screen of Khaos Kitchen.

The front-end was of extreme importance to the team in regards to user enjoyment and a large amount of time and effort was put into creating a scene that was lively and unique. Using the Maya's support for Unity, it was possible to easily import the kitchen object into the project and create a Unity prefab. This allowed for easy use of the Maya models throughout the development of Khaos Kitchen.

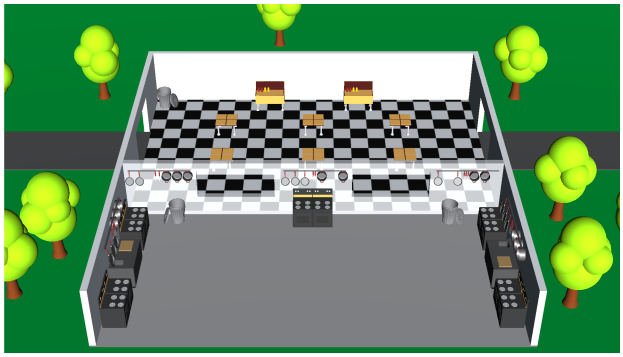

Figure 6: Kitchen Prefab

7.3.3 Chef and Customer Design

The prefab capability within Unity became even more useful for the second area in which Maya was used - the design and modelling of the chef and customer characters. The team decided that players should have the opportunity to customise their chef avatar throughout a game of Khaos Kitchen to give players a unique identity within the game and a personal incentive to do well. This meant that several different version of the chef prefab had to be created. It was realised the best way to achieve this was to create a 'super chef' that contained all the objects a player could purchase. By doing this, it was possible to enable and disable particular parts of the object in Unity as they were needed during a game. The prefab instance of a super chef allowed for this to occur. It made it able to access particular parts of the objects polygons easily. Figure 7 shows a selection of the chef variations that can be earned by players during the game.

On top of chef design the group wanted the customers to have a sense of identity. Each customer has a selection of materials to allow for some sense of uniqueness between each of them. On top of this, the customers movement will depend on the success of the restaurant. Higher performing restaurants will cause more customers to flock in, whilst poor performing will make them head to the exits.

7.3.4 Prefab Animations

Once the chefs and customers were designed and placed into the game scene, it was decided that to add to the flow, immersiveness and entertainment of the game, animations were required. Bringing the

Figure 7: Chef prefab with various customisations enabled

kitchen to life was essential to creating khaotic and engaging visuals. Adding these simple yet effective animations throughout the game; from a spinning logo and pulsating buttons on the home screen all the way through to the chef cheering animation on round completion, have allowed for the whole game to come alive. This really added another level of depth to Khaos Kitchen. A list of animations in game include; spinning logo, pulsing buttons, infinitely scrolling background, chef and customer idle animations, bouncing animations and cheering animations. The animations for prefabs and Maya models were connected together via Unity's inbuilt Animator. This allowed for animations to change on different triggers from Boolean changes to certain game events such as completing a round. The animations for the UI was achieved through object transforms and `MonoBehaviour` scripts attached to the objects.

Using Unity's particle effects system, fire prefabs are spawned in the virtual kitchen when the players make mistakes. These fires are impressive looking and are a very clear visual signal that things are going wrong for the players.

7.3.5 Animating the UI

In designing the User Interface, it was decided to work on the ideas of clarity and engagement. The player should never have issues interpreting what is required of them or the information presented to them. Another design decision was that at no point should the user's screen be static.

To achieve this a scrolling background was used by slowly translating a series of large panel objects and resetting once out of the cameras view, giving the illusion of an endlessly scrolling background. The buttons for player actions were also given idle animations where they stretch vertically and horizontally alternately. To ensure this did not inhibit the players ability to read the buttons, the text was kept at a constant location and in a clear to read colour scheme, black text on a white background.

7.4 Gameplay

When it came to the gameplay mechanics for Khaos Kitchen, there were various decisions to be made. The group wanted to structure the architecture of the code such that it achieved the desired results in the simplest and most efficient manner. The gameplay had to balance responsiveness and efficiency with the fast pace nature of Khaos Kitchen.

7.4.1 Game Architecture

The game architecture is based on interactions between scripts. The scripts behave as follows:

7.4.1.1 Game Controller The `GameController` script handles the overarching control of the game. It dictates the logic for each round, including handling the start and end of the game, the start and end of round, as well as acting as the communication between the players. It is also responsible for communicating game info onto the home screen to keep users updated with game-state info through the `AnimationController`.

The server `GameController` holds a `GameState` object which is updated with game and player info every round.

7.4.1.2 Instruction Controller The `InstructionController` handles the generation and distribution of instructions and actions to and from the players, as well as determining if player actions are

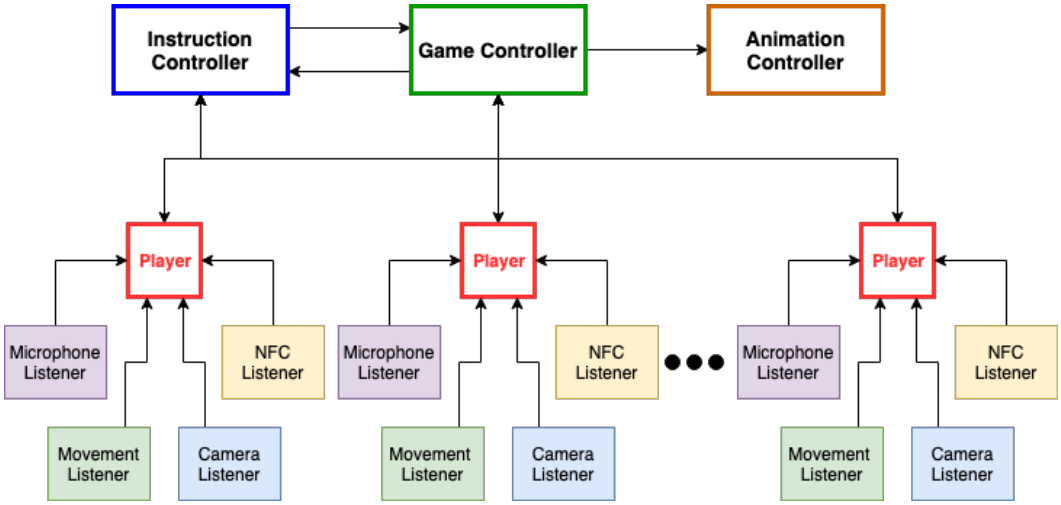

Figure 8: Gameplay Architecture Diagram

correct or not. The server `InstructionController` holds an object holding all information on player instructions and actions.

7.4.1.3 Animation Controller The `AnimationController` handles any spawning of the customers and chefs into the kitchen scene, whether that be at the start of each round or within rounds, as well as the creation and destruction of fires. Random number generators are used to create unique customer outfits, fire locations and customer movement paths to add to the khaos of the game.

7.4.1.4 Player The `Player` script handles communication data to the player UI, as well as handling data from the relevant phone sensors. It handles the user actions, whether individual or group, and outputs the results to either `GameController` or `InstructionController` where applicable.

7.4.1.5 Listeners These scripts monitor the relevant phone sensors for significant activity and sets variables in the script based off this. The Player script checks these variables and acts accordingly if it is relevant to a current instruction. These scripts are only activated when the relevant instruction has been asked of the player, preventing time outs of the sensor in use.

7.4.2 Difficulty Algorithms

When producing an algorithm to determine the difficulty curve of the game, 2 key variables were identified that controlled this; the number of instructions per round and time per instruction. It was also found through testing that the game was easier with a smaller number of players. The aim when creating an algorithm to control these two variables was to keep it as flexible as possible, such that through testing the optimal difficulty curve could be found. The equations devised were:

$$InstructionNumber = \left\lfloor \left(Base_{IN} + Increase_{IN} * (RoundNumber - 1)\right) * \left(\frac{12 - (PlayerCount - 2)}{12}\right) \right\rfloor$$

$$\begin{aligned} InstructionTime = \max\{&\big((Base_{IT} - (RoundNumber - 1) * Reduction_{IT} \\ &+ (PlayerCount - 2) * Increase_{IT}\big), \\ &MinimumInstructionTime\} \end{aligned}$$

Where variables on the right-hand side of the equation can be altered at runtime through settings on the host. The multiplication at the end of Instruction Number means a game of 8 players will receive 50% of the instructions that 2 players would receive. This is as communication of instructions is intentionally

a large bottleneck in game progress; with more players more communication is required. This creates a fair and enjoyable experience indifferent of the number of players.

The time per instruction similarly has the option to increase depending on the number of players. By setting $Increse_{IT}$ a fixed amount of extra time is added for each player more than 2. In later rounds the time per instruction may become infeasibly low. To amend this a $MinimimumInstructionTime$ is used to set a lower bound on this variable to ensure instructions do not become impossible to complete in the given time but continue to be challenging.

7.5 User Experience

Conceptually, the game mechanics are simple. Completing actions when instructed to, with no action being particularly complicated, requires very little understanding. The complexity of the game arises from inter-user communication; success is dependant on instructions being clearly communicated to the whole group. This requires a mindset unlike most multiplayer mobile games.

The round cycle is repetitive such that after one or two rounds most users will have complete gameplay understanding. However as instructions are randomly generated the repetition is not tiresome.

Colour aids players in identifying their race location; Symbols provide in-game assistance for the physical actions required of the user. Sound effects, haptic feedback and a green and red colour system aids users to gauge if their actions result in success.

This clear, uncluttered UI ensures users to not get confused while attempting to complete instructions.

The final assistance for user experience is a succinct tutorial. It involves annotated screenshots walking the user through basic gameplay, phone interactions and group activities. It also will involves a simple explanation of how to get the phone to efficiently interact with the provided NFC tags.

All of these features should allow the user to feel proficient having played the game a few times.

7.6 Games Day Set Up

To give the users even more of an immersive experience of a Khaos Kitchen on Games Day the room was to be transformed with various props so that the feeling of working a real world kitchen was being achieved. This included a mini-fridge, food items, cutlery and crockery. It was then possible to use these kitchen utensils and tie them in with the instructions users had to complete. For example an NFC instruction could be to “grab an apple from the fridge” and the user would have to open the physical fridge to tap their phone on an NFC beacons next to an apple.

For games day, specific NFC beacons were produced. The NFC scanning speed was not particularly fast at first. After some research, two methods to speed this up were devised. Putting the NFC sticker on a hard angled surface with some give, shown in figure 8. This consisted of two laser cut plywood panels attached by a sponge. This decreased average scan time as this would keep the sticker perfectly flat. This is important as even the smallest fold in the tag could stop it scanning entirely. The angled surface helped scanning as being as close to the tap without scanning was ideal. The angle achieved this as the users naturally put the top of the phone against the wood leaving the camera, where the NFC scanner is located, a very small distance away without touching. . The second was to use a larger, but not largest, 37mm NFC tag. This was found to be the fastest after trying a variety of sizes. A lattice of multiple tags overlapping by various amounts was tested however just one tag was found to be the fastest in the end likely due to interfering tags.

Figure 9: Prototype NFC beacon

7.7 Team Problems

Various technical challenges were encountered throughout the development of Khaos Kitchen and each was dealt with via its own unique solution.

7.7.1 NFC

It was quickly realised that the phones available to the team were unable to read the same NFC tag twice in a row. This was due to the phones retaining the unique tag ID and so not being able to detect the same one again. As a solution the gameplay was changed so that players never get sent to the same NFC tag twice.

7.7.2 Limited resources

An issue throughout development of Khaos Kitchen was the limit of only being able to build the game on four phones. The phones were not received until late into term one, which resulted in the group being confined to laptop testing. Many errors were only caught through testing with an Android device, however, with six members of the group, but only four phones, Android testing and development was congested.

Once these phones were split up it meant that some members of the team didn't have access to a phone which made testing of their work very challenging as the code didn't always perform on the phones as it did on laptops.

One of the phones was far less responsive to the NFC tags. This has resulted in reluctance to use it for testing and use during Games Day.

8 Third party contributions

- Sound effects - https://www.zapsplat.com/sound-effect-categories.
- Composer: Constantine Gaziotis.
- Although almost all of the models were designed in house, the fire prefab and the tree prefabs in our Kitchen scene were taken from Unity Asset Store and turbosquid.com respectively.
- All of the fonts used were from the open source Google Fonts.
- Special Thanks to Gordon Ramsey.