The Unity Game

The Final Report

The Space Racer

Project manager: Silvia Balogova

Developers: Scott Davidson
Peter Henry
Cormac Reid
Conner Turner
Lewis Wood

Executive summary

We have developed “Space Racer”, a racing game with a retro-futuristic space theme. The game concept takes inspiration from the Wipeout game series in that it shares the hovering mechanic and has a handling model that leans more towards the “Arcade” style. This means that it has the potential to appeal to a wider audience. There is still plenty of depth and room for skill and technique development to keep those who prefer a more challenging experience entertained. Also akin to Wipeout, the game takes place on closed off linear tracks. These are created using pre-made sections which increases expandability as the sections can be re-used multiple times. It also allows the track to be changed more easily than if we had created one big track model. The track uses a holographic shader to give the game a retro-futuristic theme.

The game is developed for Microsoft Windows, but the utilisation of the Unity game engine means that in the future it would be easy to port it to other platforms such as PS4 and Xbox One. We had some important requirements for the game, such as having a time trial objective and a space theme. We also had some additional features which we managed to add with the time allocated, such as controller support and the creation of atmospheric trackside objects to make the level more interesting. These, alongside some ideas that were not implemented, are elaborated on in the “Delivered product” section.

The project generally went well, with us all being assigned roles right at the beginning which we were all happy with through to the end. We used Trello to document the tasks and their completion, and Discord to communicate with each other directly. We also met weekly to discuss what we had achieved in the previous week and agree on the tasks for the following week. More on this topic is discussed in the “Closing audit” section.

We learned a lot about working in a team during this project, such as the importance of having face to face meetings as opposed to communicating solely on virtual platforms. We also learned more about the tools we were using, such as the issues with Unity’s own version control and problems with transferring models from Blender to Unity. These are explored in more detail in the “Lessons learned” section.

Throughout the duration of the project, we were fortunate enough to not have any issues with attainment. All members contributed a similar amount, and nobody was considered to have consistently failed to keep up with the work assigned to them. As a result, the project went smoothly and there was little conflict between team members. The spreadsheet detailing our contributions for each week of the project is shown in the “Team contribution” segment.

Finally, in the appendices section, there are a few pictures of the delivered product, showing the graphical style and technical achievement of our project. The appendices also include our Moscow prioritisation, some elements from our PMIS and the coding standards document.

Delivered product

Our finished product is a game, called “Space Racer”, which was developed for Microsoft Windows using the Unity engine. In the game, the player pilots a spaceship around a pre-built track, competing against the clock. We agreed that we must make one polished level and focus on making sure the ship was behaving exactly the way we wanted it to.

To complete the demo we wanted, we came up with a list of requirements for the game using the MoSCoW format (Must, should, could and won’t). Firstly, our musts include the game launching to a main menu rather than just the game, as this is how most games work. We also developed working controls for the game so that the player can control the ship. Another basic requirement was the ability to move the ship around the track. For our game mode, we decided that a time trial should be the first mode we implemented, as it was both the most basic game mode for a racing game, and it was simple to implement. As the theme of the game is space, we created a space-themed environment for the level. For our demo track, we made a brightly-coloured, semi-transparent track within orbit of a planet. Finally, we added our own custom music to improve the atmosphere of the game.

On top of this, we came up with some additional features that we could add in to the demo if we had time. We developed a strafing mechanic to allow the space ship to move side to side so that skilled players could take corners faster. We also added a boost mechanic in the game as this is a common feature in arcade racing games. In addition to these features, we implemented light trails and particle effects to the ship to further immerse the player in the game and increase the sensation of speed. In conjunction with this, we also managed to get the ship’s engine to make noise with the pitch based on how fast the player is going in the game, to add to the feeling of speed. Since playing racing games on a keyboard and mouse is regarded to be a suboptimal experience, we added support for both Xbox and PlayStation controllers. This was also done as in the videogame market, there are multiple platforms that players use, adding support for both controller types would make it easier to port the game to the Xbox One and PlayStation 4. Later in development, we decided that in order to make the level feel more real, we should add trackside objects and decorations to the game. The final product we delivered met all our musts for the project: all the additional features outlined above, a complete library of track assets that we could use in the future, and multiple different track models.

We did have some other ideas that were not implemented into the game. These ideas were unfortunately not included in the final submission, but we would have liked to have included them if we had more time. We wanted to add support for virtual reality headsets as we thought it would be an interesting addition to the game. We decided that this would not have been a good idea due to the fast-paced nature of the game giving a high risk of motion sickness when playing. We also considered adding a first-person cockpit camera, but again, we were concerned by the risk of motion sickness. We also considered adding dynamic music that would change based on where the player was in the level. Unfortunately, we did not have enough time to implement this. Finally, we wanted to add dynamic obstacles in to the track so that each lap would be different. We realised that this might not be fair for the time trial game mode, but if we had been working on other game modes, we would have considered adding this feature.

Closing audit

The team started with formulating the scope and requirements of the project. The MoSCoW list prioritisation was used here to agree on deliverables according to their importance. The team appointed the roles of project manager, programmers, artist, modeler and level designer based on our strengths.

We created a Project Management Information System (PMIS) with Project Summary, Business Case, and Configuration Map to create a product-based understanding for the project. The team decided on tasks, activities and deadlines for the Schedule Model to see the timeline. It also provided a visual demonstration of tasks duration and dependencies. We kept everything simple and clear as the specific details and activities of the project would be decided at weekly sprint meetings.

We further set up an RIC Register to keep track of risks and issues, and a Project Health Register to monitor the health of the project and team satisfaction on a weekly basis. The team satisfaction was especially important to us and even though it fluctuated throughout the project by 10%, the average satisfaction was around 87%. The Project Health Register also contains all evaluation such as the Preparation Audit and the Weekly Audit information. The early weeks' values for Weekly Audit were much lower than the rest with an average of 50.3% but they improved from week 4 with a median of 92%.

A OneDrive folder was set up to keep all the documentation organised in one place and always accessible to all members. We created a Trello board with the list of our main tasks of MoSCoW list, To-Do, Doing, and Done lists.

We held face to face weekly group project meetings with a clear agenda written beforehand to discuss progress from previous weeks where the team members' effort was marked in the Contribution Sheet. We further talked about any issues with technical side of the project. When there were technical difficulties that could have a significant impact on the main deliverables of the project, we took time to examine them and address them appropriately, whether it being dedicating more resources, or changing or excluding parts of the project. We also used our meetings to assign new tasks for each member for the upcoming week. The fair tasks division was difficult, but we find that honesty and enthusiasm motivated us to set high goals for every member for each week. We took minutes to remember what was discussed and these were available to look at any time.

After every group project meeting the project manager created new tasks in To-Do list in Trello, assigned the members to the tasks and set due dates. The Trello tasks' comments we used to communicate about the tasks that were being worked on. We used a Discord server to organise all meetings, assignments and all other communication relevant to the project.

The team agreed that project was overall well managed and working on it was enjoyable. The motivation to deliver the best project possible and consistent work proved to be sufficient in keeping the team working hard.

Lessons learned

Informality in Group Meetings

The weekly meetings that were held with our entire group were mostly of an informal nature. While this benefited us and allowed us to work in a more relaxed atmosphere, it sometimes led to us getting off topic. We found that overly formal meetings would result in people becoming bored and losing focus which could result in little meaningful work being completed. This led us to an informal approach which meant more relaxed meetings, with the group feeling happier and having better overall productivity.

Actual Face to Face Meetings

Throughout the course of our project we decided it was best to have regular face to face meetings. We found that this form of meeting had more benefits than other types such as a group calls or messaging. As the project manager organised the supervisor and team meetings one after the other, we could ensure all members were present. With all members present, we were able to easily discuss the work done the previous week, any issues we encountered, our plans for the coming week and any new ideas or changes to the project if we had any. We found that having meetings in person ensured that more work was done compared to a group call because it allowed us to physically demonstrate the work that was done, and it was easier to present ideas. It also improved the attention of the group members, as when talking online some people were less likely to concentrate on what was being said due to external distractions.

Usefulness of Trello

To keep track of our progress throughout the project we used Trello. Trello is a platform that allowed us to create multiple lists which helped us to keep track of our individual tasks that had been assigned at group meetings. It was beneficial to us as it let us record a list of tasks that were not being worked on, tasks currently being done and those that had already been completed. Another feature of Trello that was found to be useful was the ability to keep track of who was assigned to which tasks. These features allowed us to easily keep track of the state of the game and see where different members of the team were in terms of task completion.

Version Control

Since the members of the group were working mostly independently and on their own computers, a version control system was required. The game engine we used, Unity, comes with its own version control feature called "Unity Collaboration ". We used this first as it was easy to set up and it integrates well with the engine. However, issues soon presented themselves.

The first was that in order to access the project, group members had to be assigned a "seat" in the project settings. This could only be changed by the owner of the collaboration project. This meant that if the owner was busy, no changes could be made to the seating order, preventing some people from accessing the project.

The second issue was due to Unity's business model. We were using the free version, which allows up to three people on a collaboration at once. Since we had five members that would need access to the project, this compounded with the seat allocation problem, and became a big inconvenience as the project matured.

The final issue that prompted a solution, was that the collaboration feature does not give members as much control over the project as other services. If the parts of the project were changed by different members, merging these changes happened rarely without fault, and switching back to an earlier version of the project proved to be a hassle.

The solution that was proposed involved moving the project to the Git version control system, using the GitHub website to host the repository. This required us to modify the current project settings and files to support Git, though once it was moved everyone thereafter had access to the whole project. The member keeping track of the Git repository compiled a document detailing the standard for using this. This included handling issues such as conflicts with modifications from other members. Using Git, we could separate our own work, allowing a single member to merge the differences without affecting the main project, minimising the interruption to the workflow.

If we had researched version control systems before we started using Unity Collaboration, we would have started using Git from the beginning of the project and made a plan for Git and version control in general, instead of making do until it became a bigger problem.

Programmer Communication

Having multiple members working on the code base requires a certain quality of communication. For the most part, ideas and features were discussed between the two programmers. However, due to the differences in their schedules, they rarely worked on the code at the same time. This led to some problems.

One issue was code conflict. One feature was implemented and updated to the project and was later replaced by a different implementation of the same feature. Even though this feature was discussed before it was implemented, some miscommunication occurred resulting in both programmers implementing it. It proved a little difficult to fix since the behaviour of the code was very similar but the way it was written was quite different, and neither programmer wanted to mess with it further. The eventual fix was to roll back to the specific version that seemed more easily maintainable, chosen after some discussion later on.

Another issue was the difference in style between the two programmers. This was something that could cause some friction, as each programmer had a different preferred style to write code. Something that helped them keep on the same page was constructing a coding standards document (See Appendices), which detailed specific ways to write code and gave rules for them to follow. This allowed the code to be a bit more consistent and made it easier for them to read each other's work.

On occasion, a programmer would write code in bulk, leading to fewer incremental changes in the update, creating more opportunities for bugs to appear, and increasing the amount of conflicts that

could arise. Along with the previous issue of different styles, this made bug fixing a little more difficult, as there was more code to read and understand.

This was more of an issue at the start of the project, where code features were more linear, and they worked in the same area. The solution presented itself as the project matured, since there were more features to implement and each programmer would choose the features they would work on. This allowed them to work mostly independently and simultaneously without creating conflicts in the project.

Programmer communication overall could have been better, especially just corresponding more frequently about what people are doing. It improved towards the end of the project, although by then we were more aware about the lack of communication and sought to improve the quality to overcome such issues.

Model structure (template / modular)

To create a track, individual parts were created in Blender, a 3D modeling application, and then imported into Unity. From there, the pieces were stitched together to create a track. The reason for doing it this way was to allow for a modular system in which the track can be easily edited. This way of doing things would also allow pieces to be recycled later to create new tracks. There were two issues that appeared from this method.

The first was that initially there were only a small number of track pieces created. While more pieces were being made, the first section of the track was being put together, resulting in the it being more simplistic that later parts. To avoid this, either the initial part of the track could have been edited, or that part of the development could have been halted until all track pieces had been created. The other issue that came from creating modular pieces was that certain pieces did not align perfectly with one another, meaning that the barriers didn't join up correctly. Fortunately, this issue was purely aesthetic and did not affect the performance of the game. It meant that the visual quality of the game was at a lower standard than desired but there were no other issues arising from this. One possible fix for this would be to only join track pieces together that fit perfectly. This would fix barriers not being completely joined up but would also mean there would be a limit to which pieces could be used together resulting in a more repetitive track which is something that would affect gameplay and thus should be avoided. Another method to deal with the issue would be to create a track from the parts within Blender and fix any of the issues that occur with alignment. There are some issues with this, however, such as if changes to the track do need to be made, this needs to be done in Blender then the model must be reimported into Unity. Also, if the person responsible for building the track is not familiar with Blender, they would need to spend time learning how to use it beforehand.

Asset uniformity across applications

There was a small issue with how Blender and Unity use different axes, resulting in the spaceship being rotated incorrectly when imported into Unity. This subsequently caused issues with the movement of the ship. There were multiple attempts to fix the issue in Blender, but the problem remained. A fix then had to be done in Unity which involved rotating the spaceship to how it should be then putting it within an empty game object. If we had known about this issue before beginning the project, we

would have looked for alternative modelling software or maybe even consider a different game engine.

Work dependencies

At one point in the development of the game a task was given to one person that required a task given to another to be completed before the first person could begin work. This dependency resulted in one person having to wait while the other completed their work. Not having dependences is important in ensuring that the project is done efficiently and ensuring there is nobody without any tasks. Other work was also not able to get sorted so one person was left waiting for a few days while the other did their work. This problem could have been avoided if there was a check to ensure there were no dependencies when tasks were being assigned at the meetings. If a quick meeting was held to deal with the situation and the person was assigned another role there would be less time wasted.

Weekly contributions

Week	Date	Silvia Balogova	Scott Davidson	Peter Henry	Cormac Reid	Conner Turner	Lewis Wood	Total
2	21/1 - 27/1	16.7%	16.7%	16.7%	16.7%	16.7%	16.7%	100.2%
3	28/1 - 3/2	16.7%	16.7%	16.7%	16.7%	16.7%	16.7%	100.2%
4	4/2 - 10/2	16.7%	16.7%	16.7%	16.7%	16.7%	16.7%	100.2%
5	11/2 - 17/2	16.7%	16.7%	16.7%	16.7%	16.7%	16.7%	100.2%
6	18/2 - 24/2	16.7%	16.7%	16.7%	16.7%	16.7%	16.7%	100.2%
7	25/2 - 3/3	18.0%	18.0%	18.0%	18.0%	18.0%	10.0%	100.0%
8	4/3 - 10/3	16.7%	16.7%	16.7%	16.7%	16.7%	16.7%	100.2%
9	11/3 - 17/3	16.7%	16.7%	16.7%	16.7%	16.7%	16.7%	100.2%
10	18/3 - 24/3	10.0%	20.0%	20.0%	10.0%	20.0%	20.0%	100.0%
11	25/3 - 31/3	16.7%	16.7%	16.7%	16.7%	16.7%	16.7%	100.2%
12	1/4 - 7/4	16.7%	16.7%	16.7%	16.7%	16.7%	16.7%	100.2%
13	8/8 - 14/4	16.7%	16.7%	16.7%	16.7%	16.7%	16.7%	100.2%
Contribution:		16.3%	17.1%	17.1%	16.3%	17.1%	16.4%	100.2%

Appendices

Coding Standard

Naming conventions

Namespace	Pascal	System.Drawing
Class, struct	Pascal	AppDomain
Enum	Pascal	ErrorLevel
Enum member	Pascal	FatalError
Private field	Camel_	listItem_
Non-private field	Pascal	MainPanel
Property	Pascal	BackColor
Method	Pascal	ToString
Local function	Pascal	FormatText
Parameter	Camel	typeName
Local variable	Camel	listOfValues

Code Layout

- Don't use braces for simple if/for/while statements (easier to read):

```
if (this)
    DoThis();
else
    DoThat();
```

- Braces should be on a new line (more readable)

```
Start()
{
    DoSomething();
}
```

- Maximum line length should be 80 (stops needlessly long lines to read)

- With long statements, factor out new variables (improves readability)

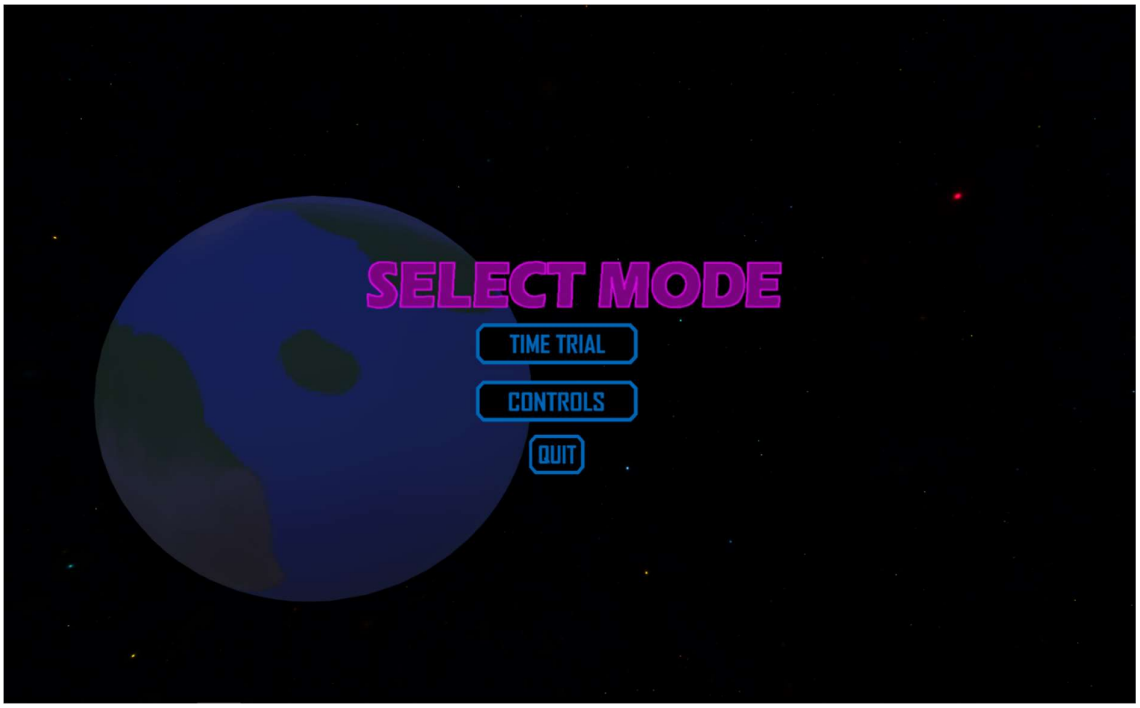
```
float clampedSpeed = Mathf.Clamp(Speed, 0.0f, TerminalVelocity);
float dragAmount = drag_ * clampedSpeed;
```

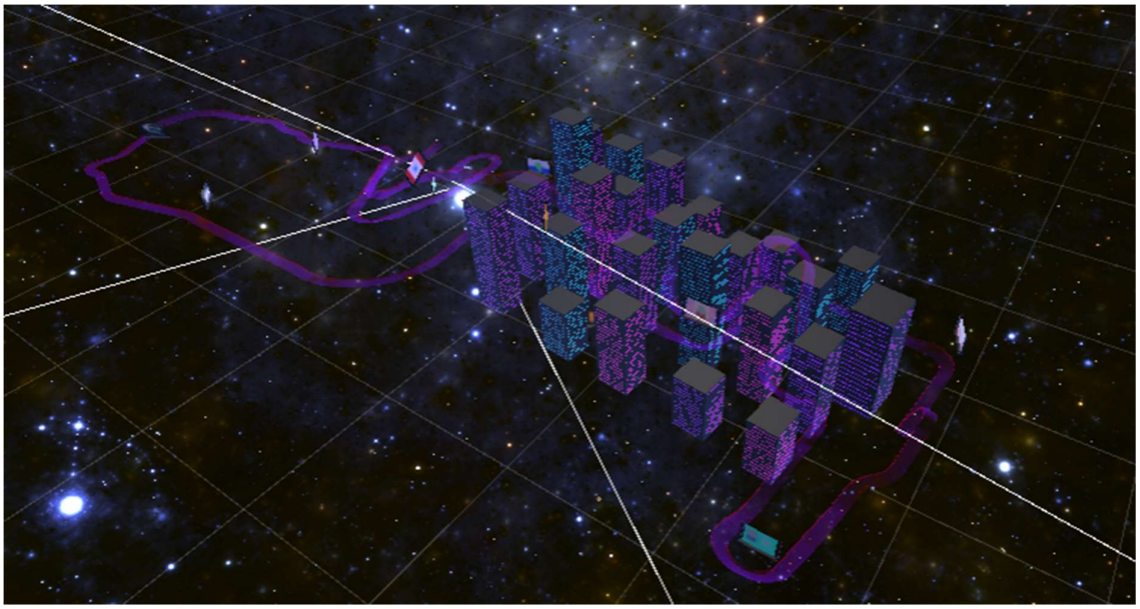
- With long function calls, insert new line at important parts

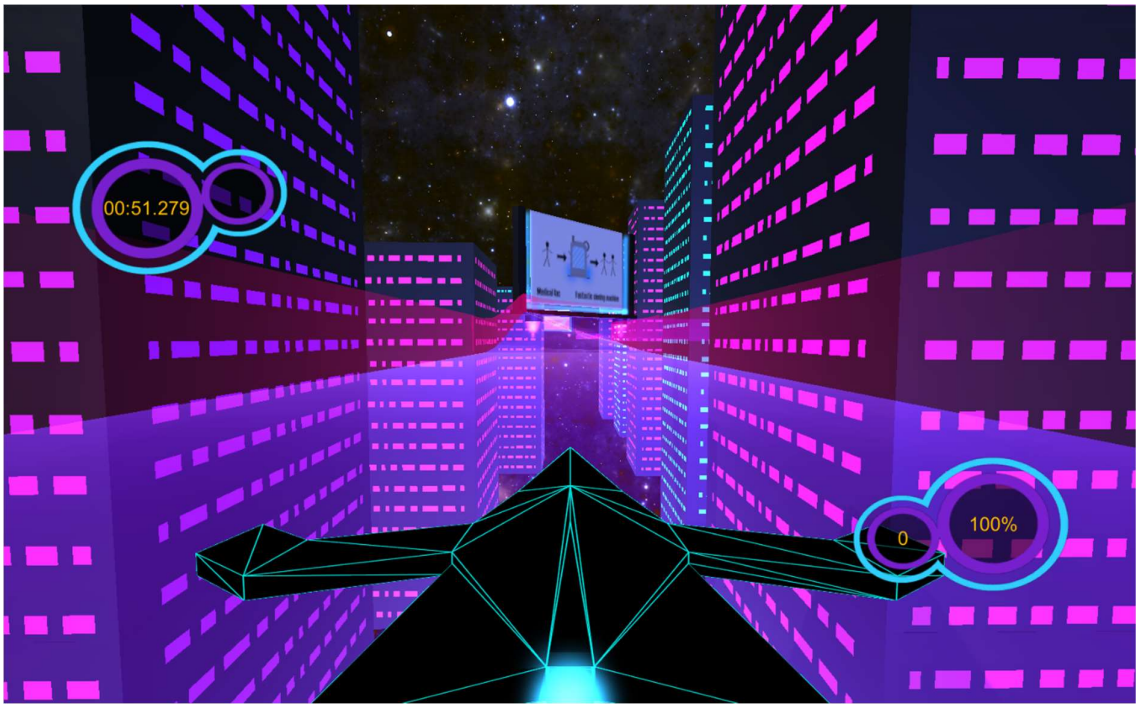
```
isOnGround_ = Physics.Raycast(ray, out rayInfo_,
    MaxGroundDistance,
    WhatIsGround);
```

- Function/variable names should describe themselves (stop comment flood)

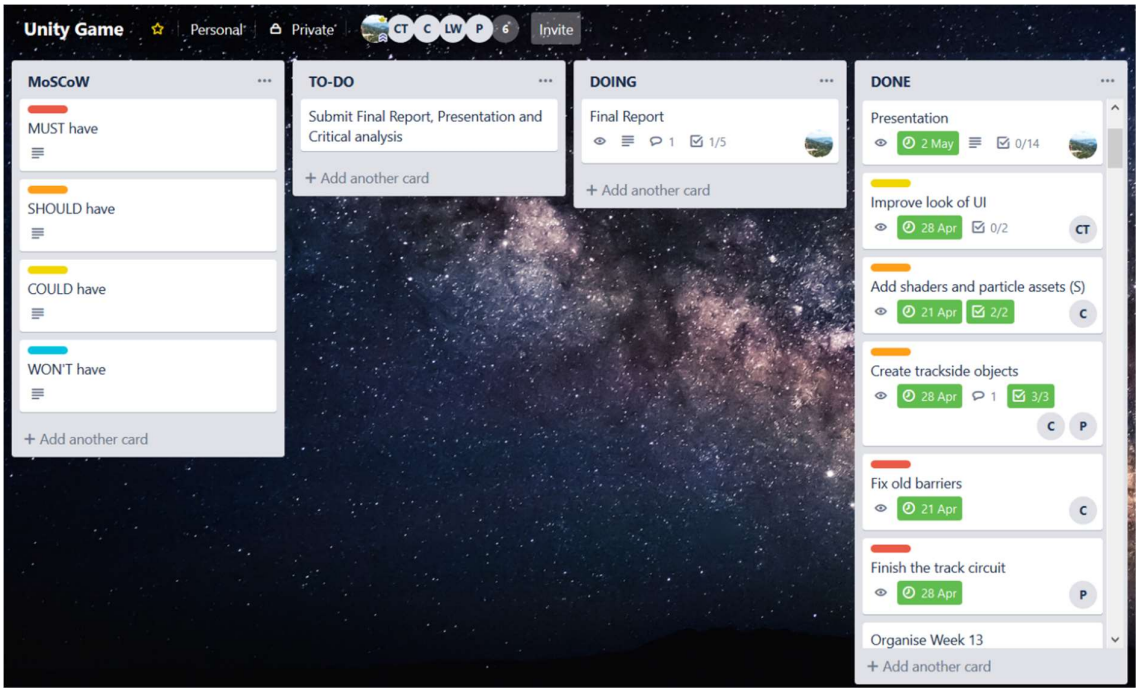
Screenshots of the game






00:51.279
0
100%

The Trello board



The MoSCoW MUST haves prioritisation list

Description Edit

Must haves:

User experience

```
- Main menu to get in to the game
- Spaceship moving around track
- Time trial goal (hotlapping)
```

Mechanics

```
- Spaceship movement
    - Acceleration
    - Turning
    - Hovering
- Collisions
    - Track
    - Barriers
```

Video

```
- Models
    - Neon spaceship
    - Modular pieces of track
- Shaders
    - Space skybox
    - Holograph track
```

Audio

```
- Music
    - In game loop
```

Project Health Documents

Preparation Audit

Weight	Item	Response
5	Is a Sponsor appointed to the project from the beginning?	100
4	Is the Sponsor involved enough in the high-level aspects of the project?	90
3	Does the Sponsor avoid getting themselves involved in the details?	90
2	Is the Sponsor familiar enough with the P2.express method?	
4	Is there a clear Charter available from the beginning?	50
5	Is the Project Manager appointed to the project from the beginning?	100
5	Is the Project Manager familiar enough with the P2.express method?	90
3	Are the Consultants clearly appointed to the project?	
3	In case you're going to have a PM Support, are they clearly assigned to the project?	
4	In case you're going to have a PM Support, are they familiar enough with the P2.express method?	
5	In case you're going to have parts of the project done with internal teams, are all Team Leaders clearly appointed to the project?	
3	In case you're going to have parts of the project done with internal teams, are all Team Leaders familiar enough with the P2.express method?	
5	Did you set up the whole PMIS?	100
5	Is it clear and easy for everyone how to access any element in the PMIS?	100
3	Did you create the Configuration in a workshop?	100
3	Are there more than 30 elements in the Configuration mind-map?	100
5	Did you identify at least 10 risks in the RICs element?	100
4	Did you use a workshop for identifying and planning risks?	95
5	Do you have effective, actionable plans for risks?	75
	Score:	92

Weekly audit

Weight	Item	W01	W02	W03	W04	W05	W06	W07	W08	W09	W10	W11	W12	W13
4	Are the progress data reliable?	30	50	60	75	95	97	95	95	95	90	95	97	95
1	Did the Project Manager add a short note to the weekly measures?													
2	Did the Sponsor check the Dashboard?													
3	Did all Team Leaders check the Dashboard?													
3	Did the Customer PM check the Dashboard?													
5	Did you create an actionable plan for recovering the deviations?	50	50	60	85	95	95	95	90	90	90	90	90	90
5	Did you have en effective weekly kick-off with relevant stakeholders?													
	Score:	41	50	60	81	95	96	95	92	92	90	92	93	92

Team satisfaction with example weeks and overall total

		Week 6						Week 7						
Weight	Item	Silvia	Scott	Peter	Cormac	Conner	Lewis	Silvia	Scott	Peter	Cormac	Conner	Lewis	Total
2	Do you know what is expected from you, and what you can expect from others?	90	88	90	100	90	100	95	90	90	90	90	95	92.5455
2	How effective is the communication between you and the person you directly report to?		96	100	100	100	100		90	100	100	100	100	98.6
2	How effective is the communication between you and your teammates?	75	72	70	80	75	70	80	80	80	80	80	80	77
4	Are the project targets set realistically?	80	84	80	90	85	90	85	75	80	90	90	85	84.9091
2	Does the project management system protect you, and help you work on the project comfortably?	95	92	100	100	95	100	100	85	100	100	100	100	97.4545
2	Is the Project Manager supportive?		100	100	100	100	100		100	100	100	100	100	100
5	Do you have a clear image of the project as a whole? Do you know your role in this mission?	90	82	80	100	85	95	95	95	85	100	90	95	91.0909
4	Overall, are you happy working in this project?	100	98	100	100	100	100	100	95	100	100	100	100	99.3636
5	Overall, how do you rate the project management system?	85	88	90	90	85	90	85	90	90	90	85	95	88.9091
0	Do you have any suggestions for improving the project management system?	v it more rhythm written down to follow												
	Score:	81	85	84	92	87	90	86	82	86	90	90	89	87.3636