Anttris

Chris Aikman Benji Cope Skyler Manzanares
Hugo Rivera Sean Turner

April 27, 2015

1 Project Overview

Anttris is a unique and competitive three-dimensional puzzle game that can be played alone or against others online. The core of Anttris' gameplay lies in solving various puzzles that are composed of blocks. These puzzles vary from simple to complicated and require critical thinking to complete. Players solve these puzzle cubes by interacting with paired and wild blocks until they are all removed.

Anttris includes two different game modes: single-player and competitive. Single-player focuses on clearing puzzles with an emphasis placed on efficiency of the solution which is measured by the amount of moves and total time. Competitive games shifts the focus to solving cubes faster than an opponent.

While the two core game modes provide a fun and unique game, Anttris really shines when it comes to the ability to create your own puzzles through a built-in editor. Players are also able to use the puzzles they created when playing competitively. The goal of the game is expanded from simply solving a cube faster than your opponent to *creating* a puzzle that will challenge your opponent while you solve their puzzle first.

1.1 Scope and Objectives

While Anttris is a simple puzzle game, it was designed to be extendable. The core scope of the game involves creating working self-contained executables for many different platforms which include – but are not limited to – PC, Max and Linux. These executables contain single-player and competitive online games modes with a working editor. To do this, we utilized a game development framework called Godot [2] that provides the basic functionality of a three-dimensional game. A mixture of networking libraries and Godot's networking features were utilized to complete the competitive game mode which was done using a peer-to-peer direct connection. Puzzles can be

both hand crafted and generated using a custom puzzle generator. To ensure that puzzles created in the editor are solvable, the game also includes a puzzle solver that quickly and efficiently determines if a generated or editor-created puzzle can be solved.

The main objectives of the project that were accomplished:

- Completed core gameplay mechanics. This involved completing a single-player game that uses user input to manipulate the game's state into winning the game.
- Completed graphical elements. This involved creating and modifying the visual elements of the game from the textures of the 3d objects to the graphical user interface.
- Completed multiplayer gameplay mechanics. This involved connecting two individual players on separate machines in order for the players to compete with each other on preset puzzles or puzzles they have made themselves.
- Completed a puzzle generator. This involved creating an efficient way of generating unique puzzles that are both fun and solvable with varying levels of difficulty.
- Completed a puzzle solver. This involved creating an efficient way of solving any puzzle, generated or editor-created.
- Added additional gameplay mechanics including non-cube-shapes puzzles.

1.2 Supplementary Requirements

1.2.1 Interface Requirements

In Anttris, the user interface is made to be simple and intuitive so that any user will be able to quickly pick up the game and play. Menus are clear and simple and the in-game user interface is minimal so that the user can focus on gameplay. The editor follows these same rules by providing maximum functionality with the least amount of visual clutter.

Standard input methods will be used depending on the device. A standard mouse on a personal computer will cover all of the games input requirements while a touch screen will be supported on mobile devices. A keyboard (on screen or physical) will be used for text input that includes entering an IP address or entering the user's online display name.

1.2.2 Performance Requirements

Anttris will follow industry standards meaning that it needs to look visually pleasing and also run efficiently on all modern machines including common computers and mobile devices. Graphics will be professional, light and clean as usual for puzzle games. Anttris will deliver a minimum of 30 frames per second with a maximum of 60 frames per second depending on the device. Intended devices include personal computers commonly found in a traditional office space as well as modern smart phones that run on iOS or Android. Frame rates will be consistent and not choppy. Failing to meet either of these requirements is unacceptable.

2 Customer Requirements

2.1 Use-Case Diagrams

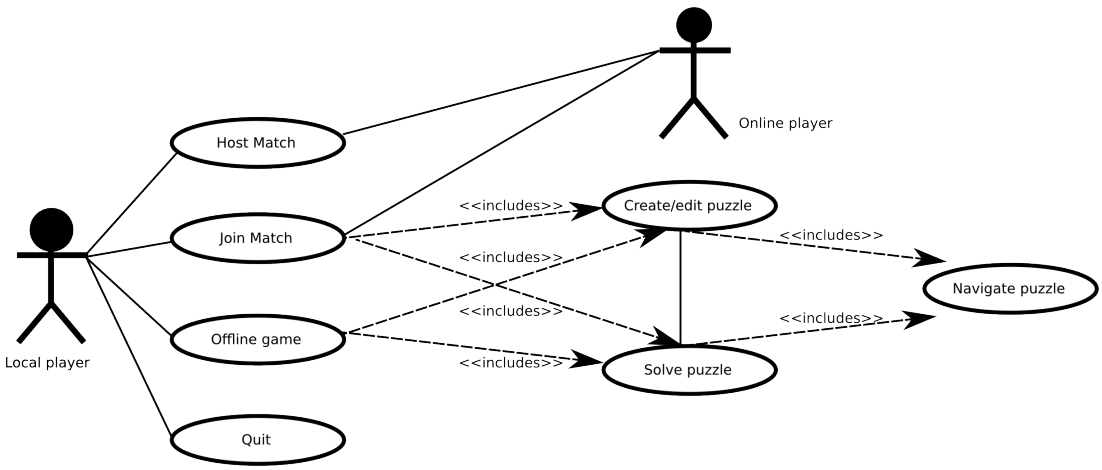

Figure 1: Use Case Diagram

2.2 Actor Descriptions

Local player is the local user and has full access to the mouse and keyboard or touchscreen interface.

Online player is a non-local player. There is two-way communication between this type of player and the local player. There may be 0 or more online players.

The Host has special powers, may disconnect other players and block players from selecting different puzzles. This player could be Local or Online.

2.3 Use-case Descriptions

2.3.1 Host match

Entry conditions Internet connection.

Exit conditions Will return to game menu. All players disconnected or all puzzles have been solved.

Participating Actors Online player and local player.

Flow of events :

1. The user presses the appropriate menu button.
2. User creates a game, selects number of players and game type
3. User shares IP address with other player. This player may connect to the match
4. Puzzle Selector appears.
5. The user may configure the game with this screen. This entails choosing the number of layers and a difficulty.
6. This Puzzle Selector will create a Puzzle Scene
7. The scene will change from the main menu to the Puzzle Scene.
8. Additional view-ports will be added to the game screen. These show the progress of the opponent by mirroring their puzzle behind and to the side of the local player's puzzle.

2.3.2 Join match

Entry conditions Internet connection. The menu must be present.

Exit conditions Will return to game menu. All players disconnected or all puzzled have been solved.

Participating Actors Online player and local player.

Flow of events :

1. The user presses the appropriate menu button.
2. This host will send a Puzzle Scene
3. The scene will change from the main menu to the Puzzle Scene.
4. Additional view-ports will be added to the game screen. These show the progress of the opponent in the same way as the Host use case.

2.3.3 Offline game

Entry conditions Menu must be present.

Exit conditions Will return to game menu. Puzzle has been solved or user wishes to return to the menu.

Participating Actors Local player.

Flow of events :

1. User selects the appropriate menu button.
2. Puzzle Selector appears.
3. The user may configure the game with this screen. This entails choosing a puzzle and its layer count and difficulty.
4. This Puzzle Selector will create a Puzzle Scene
5. The scene will change from the main menu to the Puzzle Scene.

This use case includes the Host Match, Join Match and Offline Game use cases.

2.3.4 Edit puzzle

Entry conditions A Puzzle Scene must be loaded and edit mode must be activated.

Exit conditions Will return to game menu. The puzzle may be saved to the disk.

Participating Actors Online player or local player.

Flow of events :

1. The user navigates the puzzle
2. If a position on the grid is selected, the Block Modifier is presented. This position may be empty or it may contain a block.
3. The user may change properties of the block using this screen.
4. Blocks may be added or removed using this same screen.

Puzzle preview:

1. User may press the preview button
2. The user will try the puzzle in Solve Puzzle mode until that mode's exit conditions are met.
3. The solving scene will have a button for returning to the editing scene

This use case includes the Host Match, Join Match and Offline Game use cases.

2.3.5 Solve puzzle

Entry conditions A Puzzle Scene must be loaded and solve mode must be activated.

Exit conditions Will return to game menu or puzzle editor. If the game is over, an overview of results will be shown.

Participating Actors Online player or local player.

Flow of events :

1. The user navigates the puzzle.
2. If a block is selected, the block runs any associated Block Action.
3. These Block Actions may modify the block's properties or request the removal of blocks, including the selected block, from the Grid Manager.
4. This sequence is repeated until the winning block is found or the user quits.
5. User's actions may be mirrored on an online player's screen, likewise, separate Puzzle Scenes may be updated with any moves made by other players.

2.3.6 Navigate puzzle

This use case includes the Solve puzzle and Edit puzzle use cases.

Entry conditions Puzzle scene loaded and permission to move. Input devices must be functional.

Exit conditions The game must offer continuous feedback. If the entry conditions are met, any further input must be acted on as soon as possible.

Participating Actors Online player and local player.

Flow of events :

Camera motion:

1. User drags with a mouse or touchscreen
2. The camera rotates

Block selection:

1. User clicks with mouse or taps on touchscreen
2. The 3D coordinates are translated into a position on a game "board."
3. The Grid Manager is notified of input and the grid position.
4. If a block is present there, it is selected and activated. Exact actions depend on the game mode.
5. If a block is not present, the space is selected. This is only useful in edit mode.
6. The user may end the game at any point through the pause menu.

2.3.7 Quit

Entry conditions Game must be running. This action is asynchronous and may activate at any point.

Exit conditions Game, be gone!

Participating Actors Local player.

Flow of events :

1. User presses the quit button on a menu
2. or User presses appropriate sequence of keys, such as the escape key or the alt and F4 combo.
3. Some data may be saved, such as the puzzle being currently edited.
4. The program shuts down gracefully.

3 Architectural Design

We chose the file-system to save persistent data, a single game program with both client and host modes as a networking model, and the Godot Engine for most tasks.

The file-system is the easiest and most portable method to save data. It is suited to the game's few storage requirements. We only need to save scores, custom levels and settings.

The user decides at the game menu whether to host a game or join a game, this dynamic approach is convenient for the user; a dedicated server would be too difficult for this casual game. A peer-to-peer solution would also be easy for the user, but Godot has better support for the Server/Client model.

We also considered using Unity, Ogre, Irrlicht, and Three.js for our game. All of these game engines can generate portable executables, but Godot was chosen thanks to its multi-platform development environment, unlike Unity; a complete set of libraries, unlike Irlicht and Ogre; speed of execution, unlike Three.js. Only Three.js and Godot support mobile platforms. Three.js and Unity were the closest contenders.

3.1 Subsystem Architecture

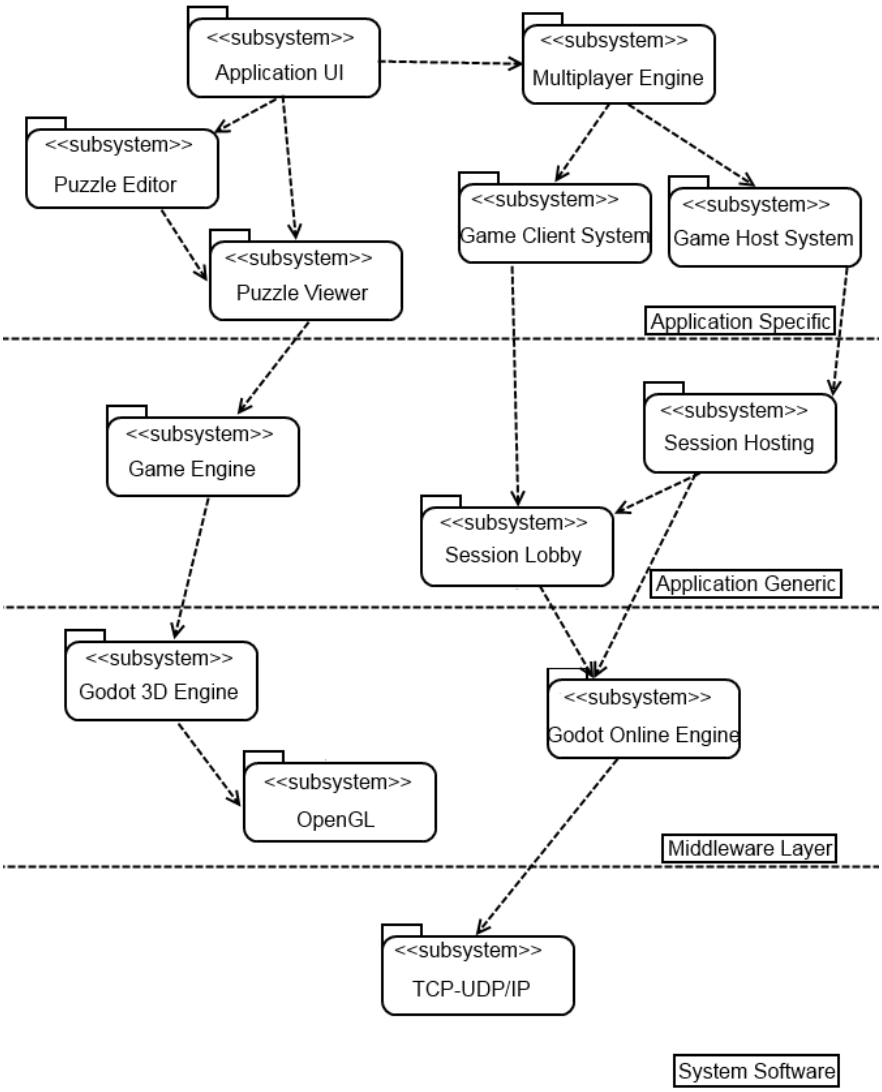

Figure 2: Layered Subsystem Architecture UML Diagram.

The two primary subsystems of Anttris are the puzzle system and the online system.

The puzzle-solving system is composed generically over a puzzle viewer and attached event handling that allows interaction with the puzzle. Both offline games and online ones use the puzzle viewer, as does the unique puzzle-editing mode. The puzzle viewer is used by all three unique modes, and works with our 3D model inter-

action system to provide interactivity. This system is built on the Godot 3d Engine, which uses OpenGL.

The online system breaks down immediately into host mode or remote client mode. The host mode interfaces with the application-generic session-host system, which works in conjunction with the session lobby to facilitate remote clients to connect and play against the host. Remote clients also work closely with the session lobby to find hosts to play against. Both the session lobby and the session-host system work with the Godot Online Subsystem to provide connectivity. The Godot Online Subsystem is built over TCP-UDP/IP.

3.2 Deployment Model

We have a simple deployment scheme: install the same game on compatible systems and allow the user to dynamically choose the session type. There will be one host machine and zero or more client machines. A game in offline mode is treated as a host machine.

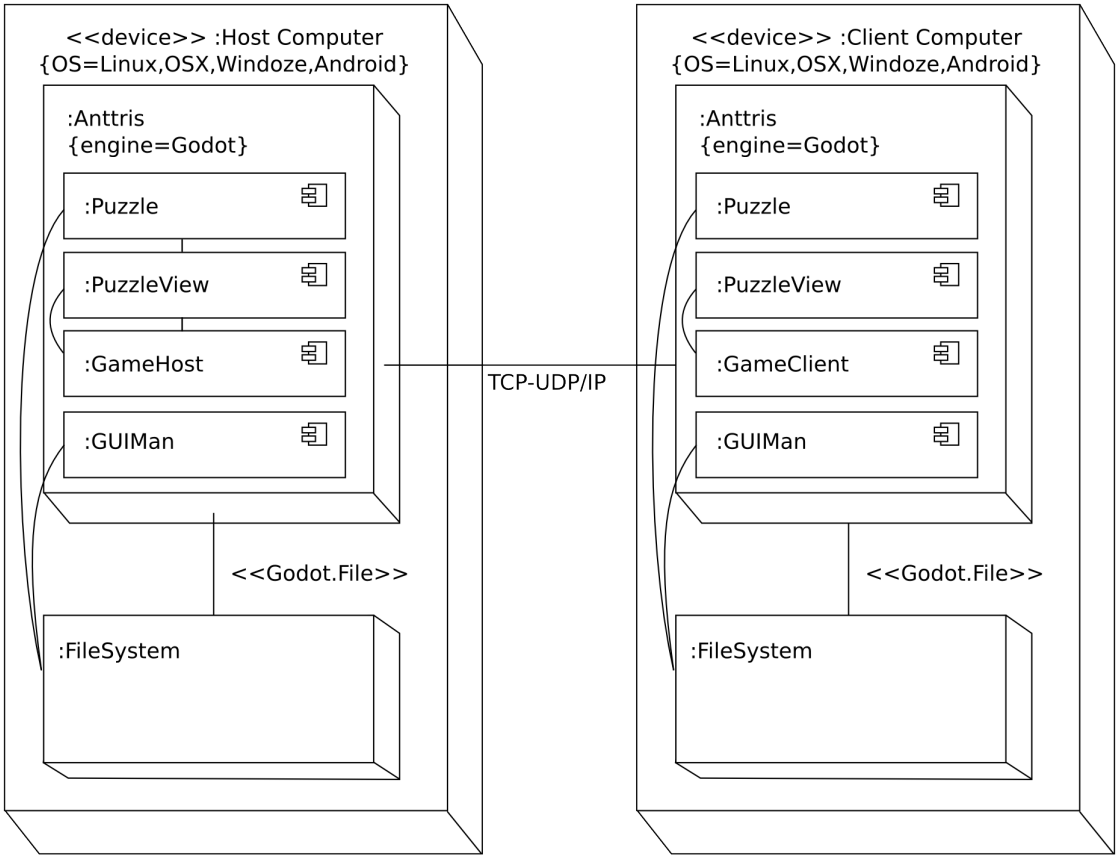

Figure 3: Deployment diagram. Installation is the same for both host and client. The type of session is chosen dynamically by the user

4 Use Case Realization Design

Every significant use case will be realized here using both class diagrams and sequence diagrams. Notice the OfflineGame use case can be modeled by a Host with no clients, no further elaboration is needed for that. The Quit use case is trivial.

4.1 Sequence Diagrams

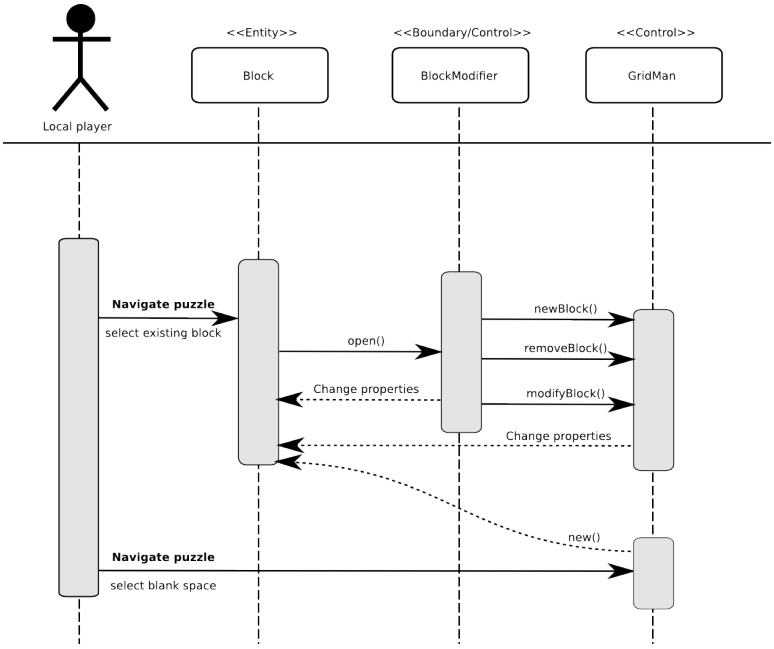

Figure 4: Sequence Diagram for editing a puzzle

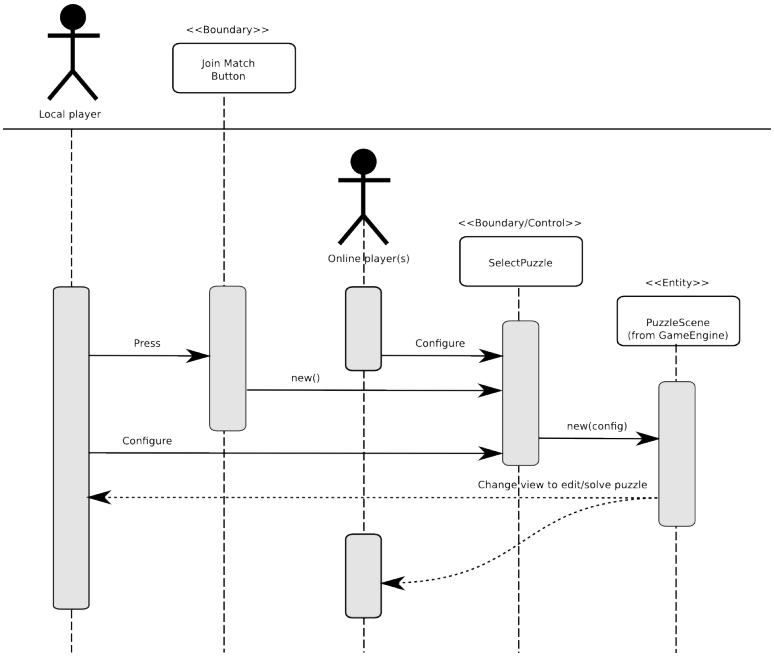

Figure 5: Sequence Diagram for joining a match

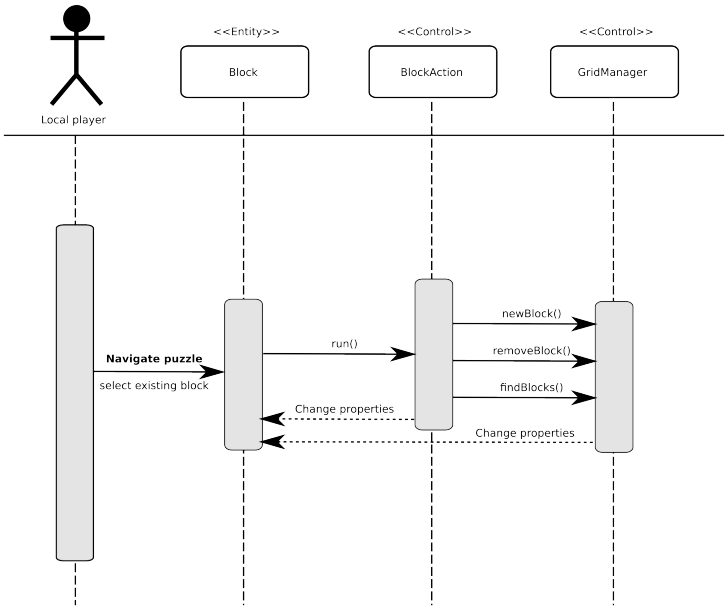

Figure 6: Sequence Diagram for solving a puzzle

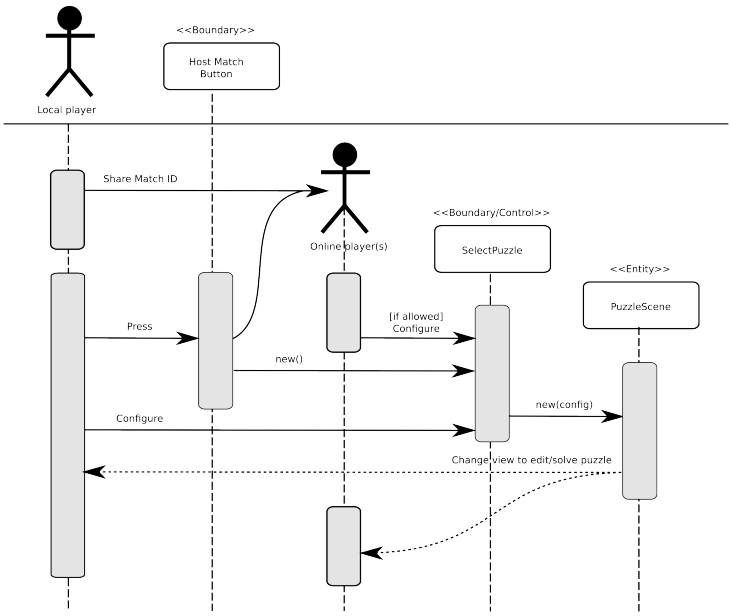

Figure 7: Sequence Diagram for hosting a match

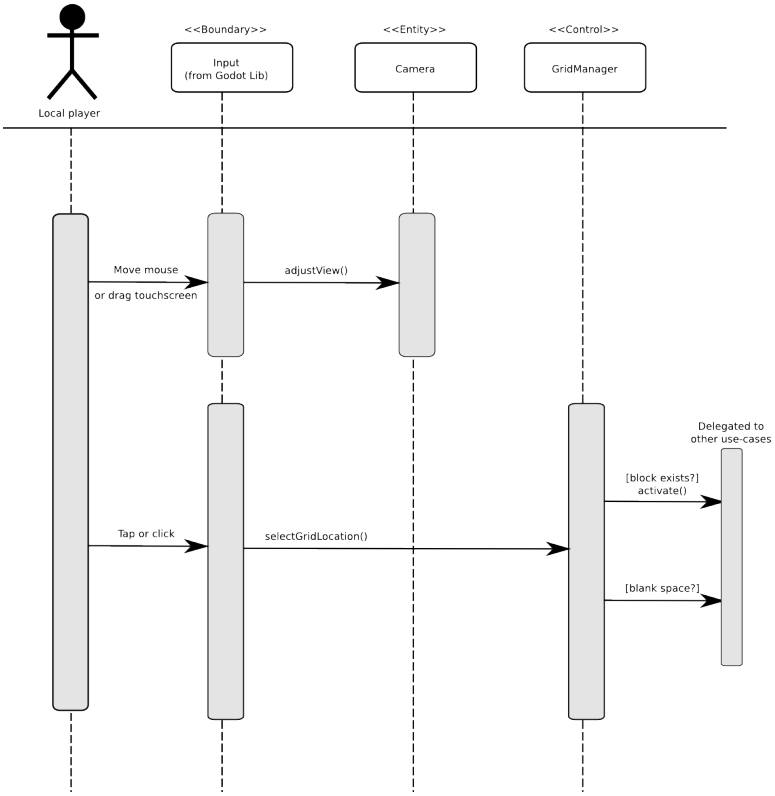

Figure 8: Sequence Diagram for navigating the puzzle

4.2 Class Diagrams

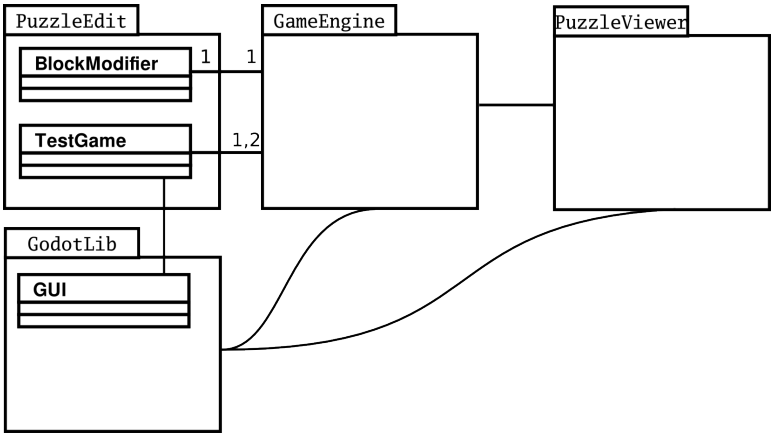

Figure 9: UML Class diagram for the puzzle editing Use Case. Notice TestGame is used to start a new GameEngine to test the current puzzle.

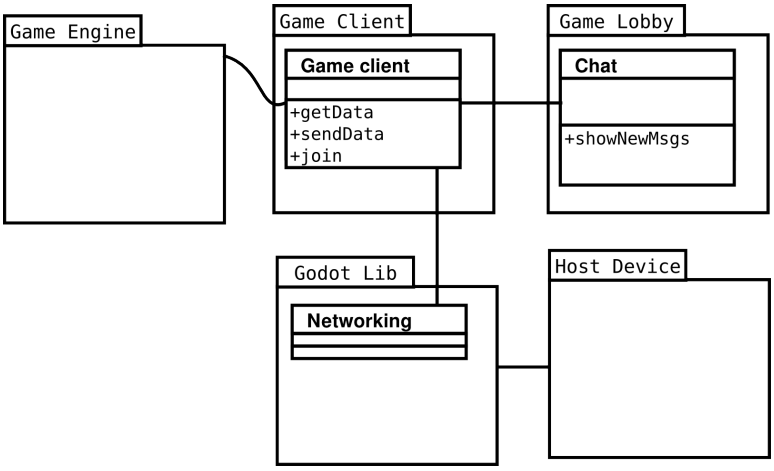

Figure 10: UML Class diagram for a client joining a game.

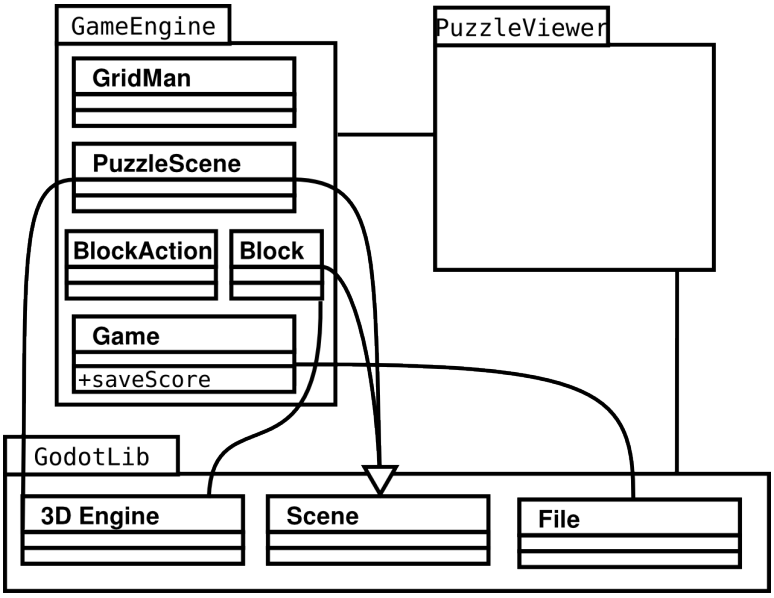

Figure 11: UML Class diagram for solving the game.

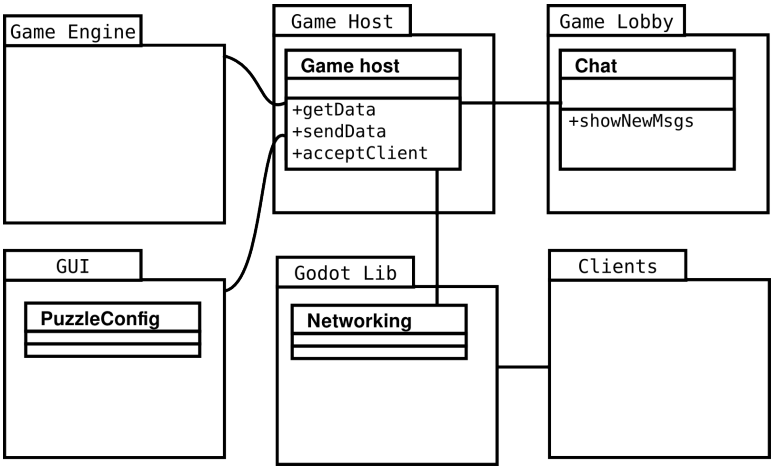

Figure 12: UML Class diagram for hosting a game.

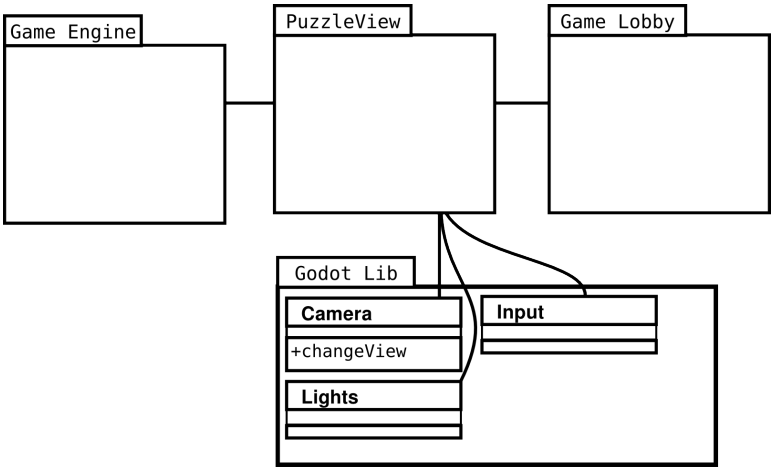

Figure 13: UML Class diagram for navigating the puzzle.

5 Subsystem Design

Anttris is broken down into the following four subsytems: GUI, Networking, File Manager and Game.

5.1 Anttris GUI

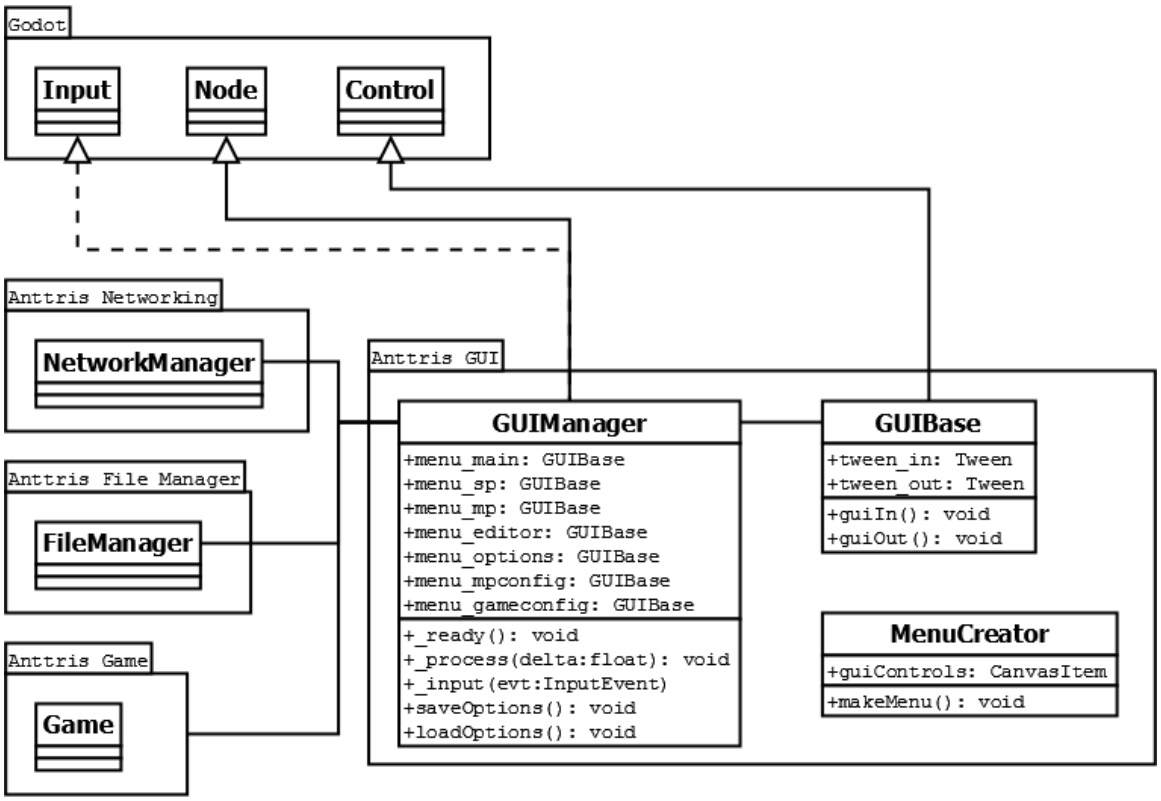

Figure 14: Class diagram for the GUI subsystem.

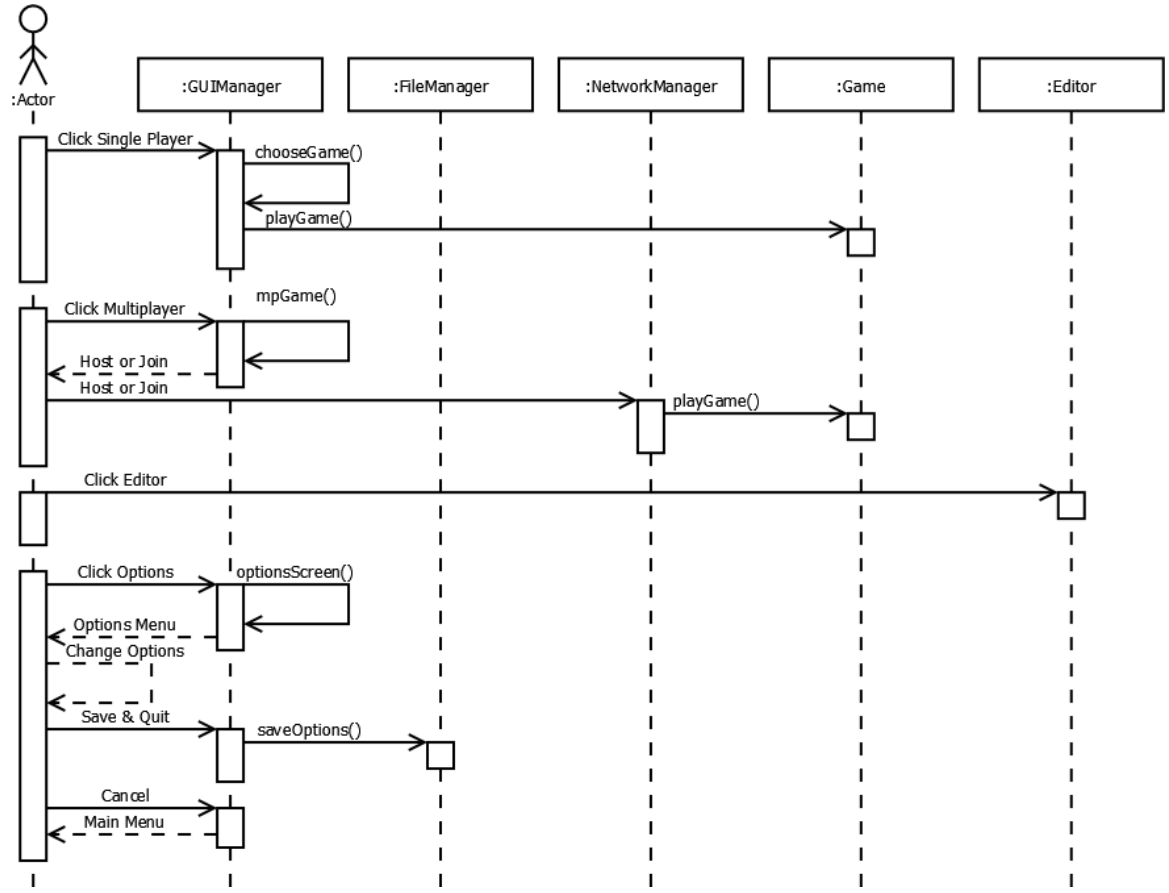

Figure 15: Sequence diagram for the GUI subsystem.

This subsystem manages all of the graphical user interfaces that make up the menus. Menus all inherit GUIBase to provide smooth and consistent fade in and slide out animations. The MenuCreator takes all of the buttons in a menu and orders them nicely for consistency. The menus themselves are instances of GUIBase.

5.2 Anttris Game Engine

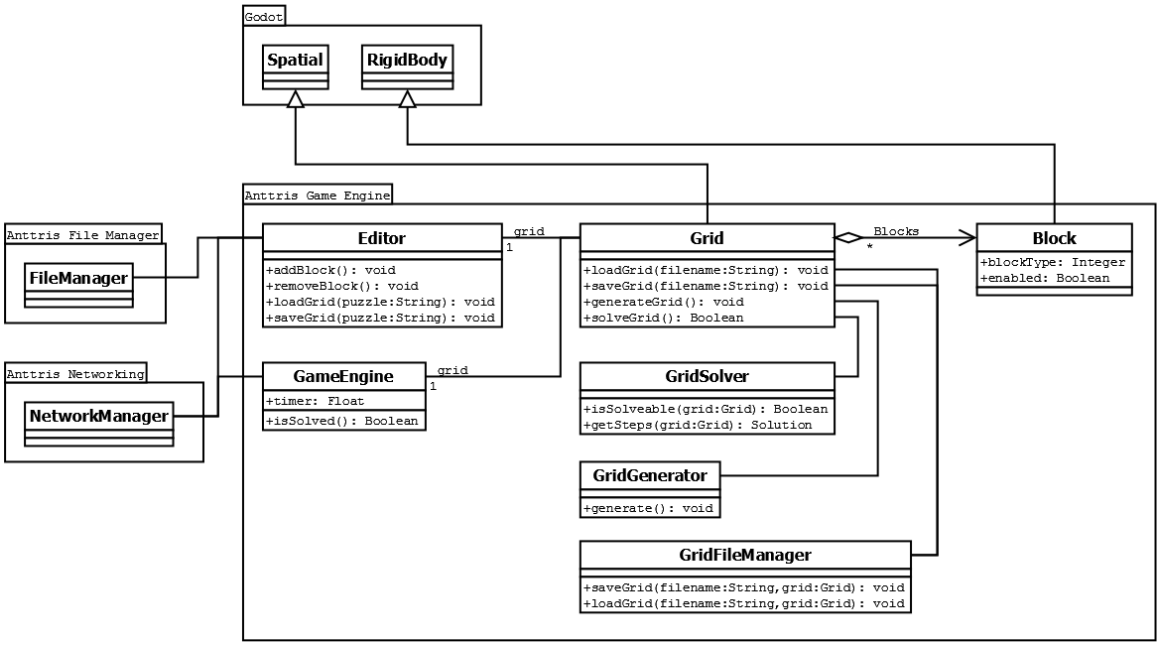

Figure 16: Class diagram for the game subsystem.

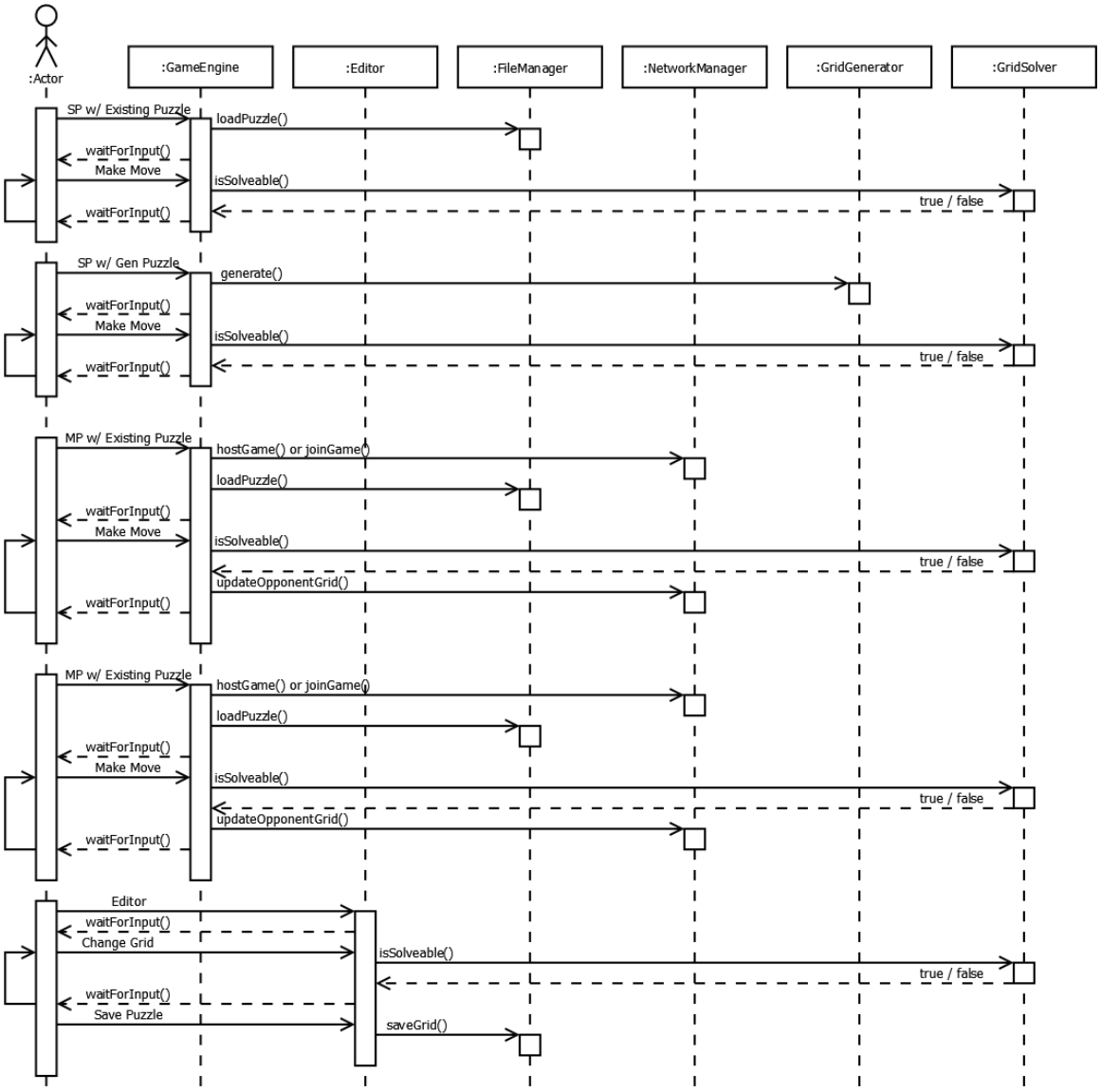

Figure 17: Sequence diagram for the game engine subsystem.

This subsystem manages all of the game and editor states. The game and editor are similar with minor differences in input. This subsystem handles both single player and competitive modes.

5.3 Anttris Network Manager

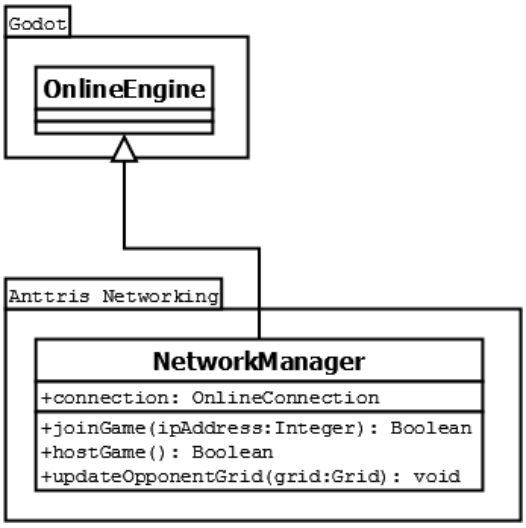

Figure 18: Class diagram for the networking subsystem.

This subsystem provides a central interface for networking. All games that are joined or hosted for competitive play will go through this subsystem. It updates the connection periodically and updates the opponents grid on the screen when they make new moves. This subsystem has no sequence diagram as it is self contained and is only used by outside subsystems. See the sequence diagrams for the GUI and game subsystems.

5.4 Anttris File Manager

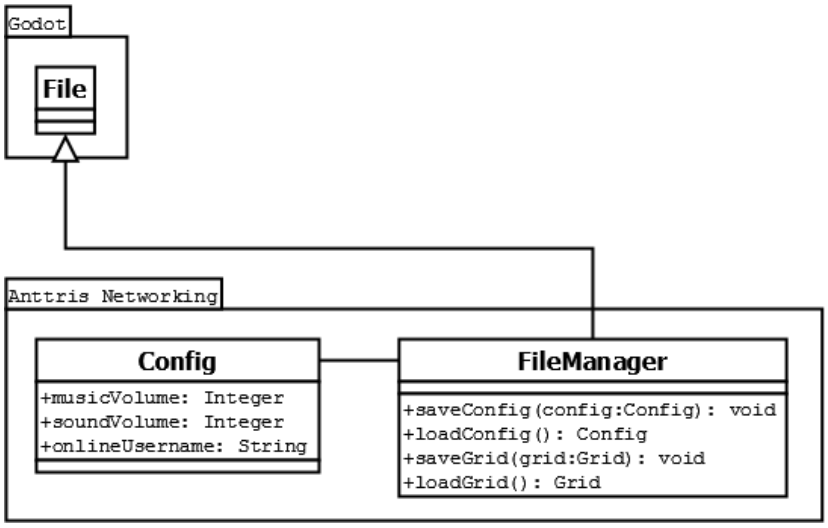

Figure 19: Class diagram for the file management system.

This subsystem provides a central interface for file management. Options are be loaded and saved from here as well as any puzzles created in the editor.

6 Human Interfaces

Screenshots included below are final in-game designs.

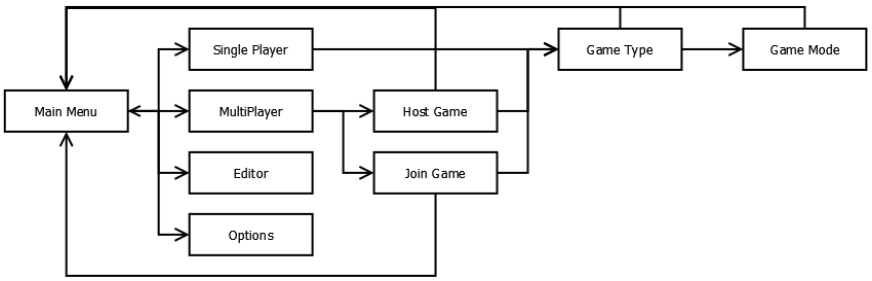

Figure 20: Overall flow of the GUI.

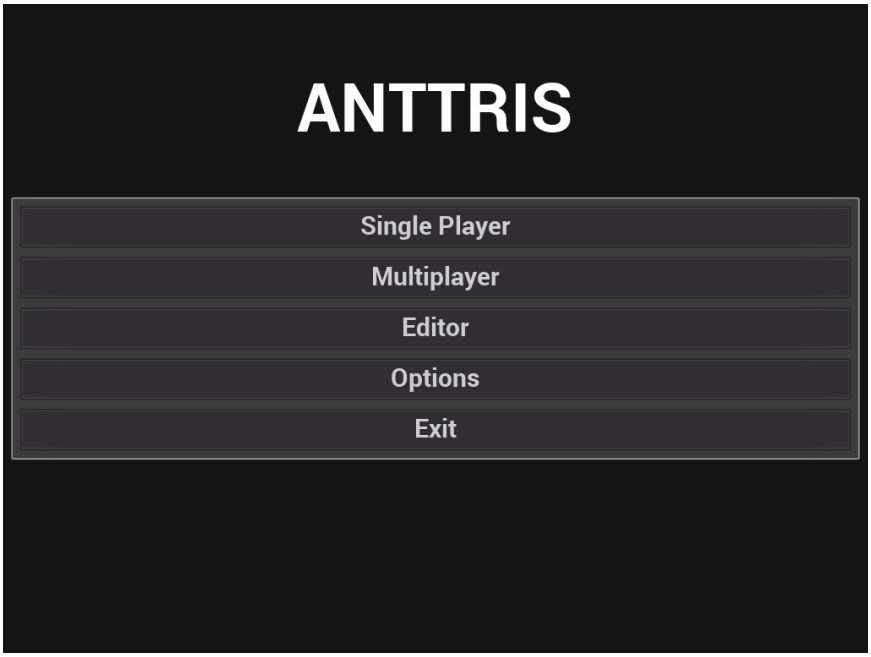

Figure 21: Main menu design.

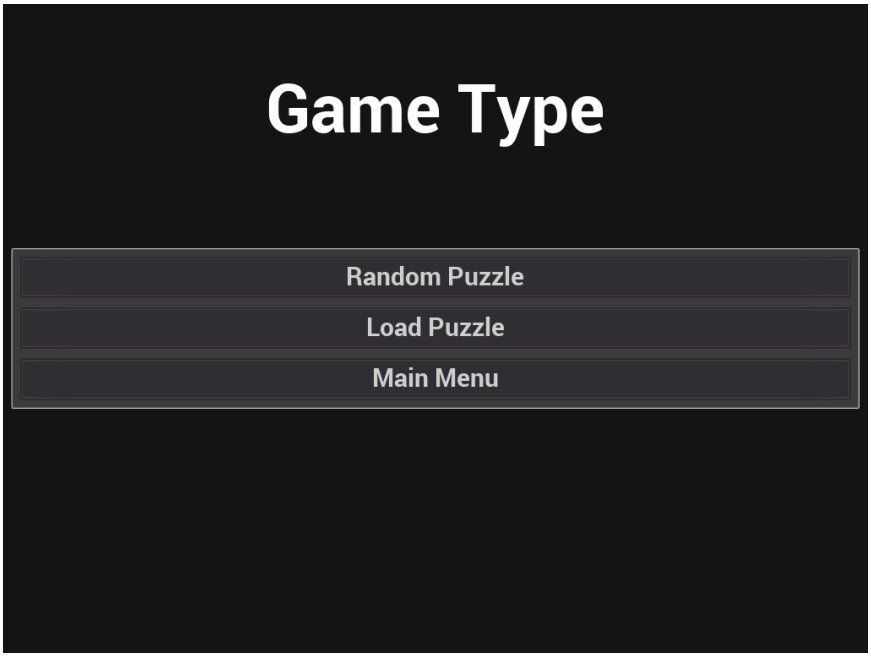

Figure 22: Game type menu design.

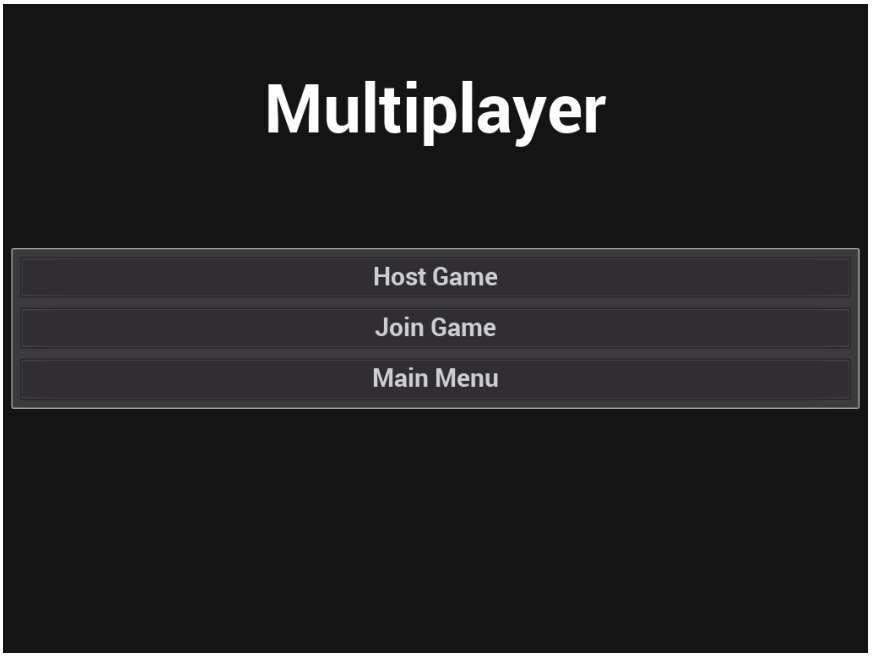

Figure 23: Multiplayer menu design.

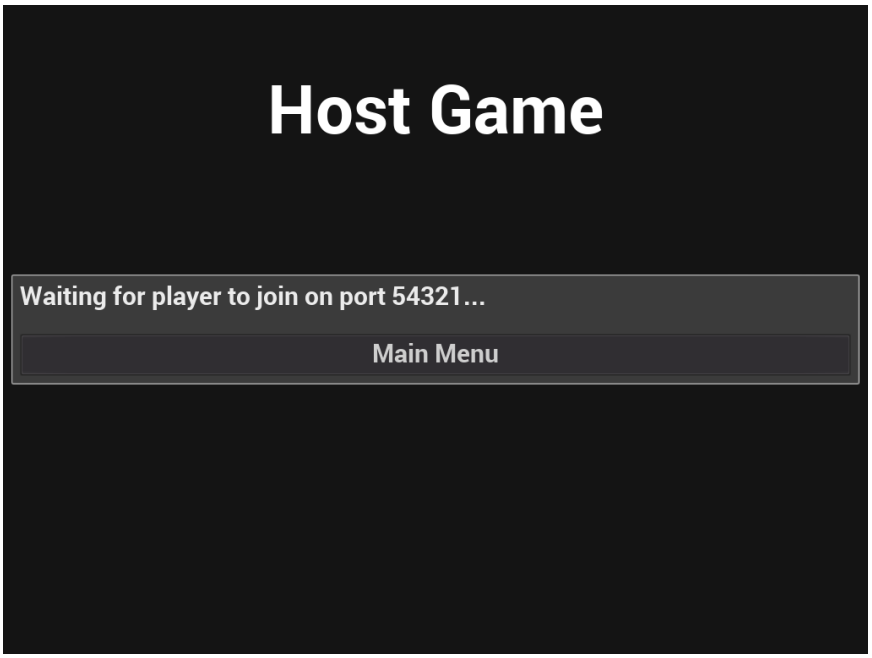

Figure 24: Host game menu design.

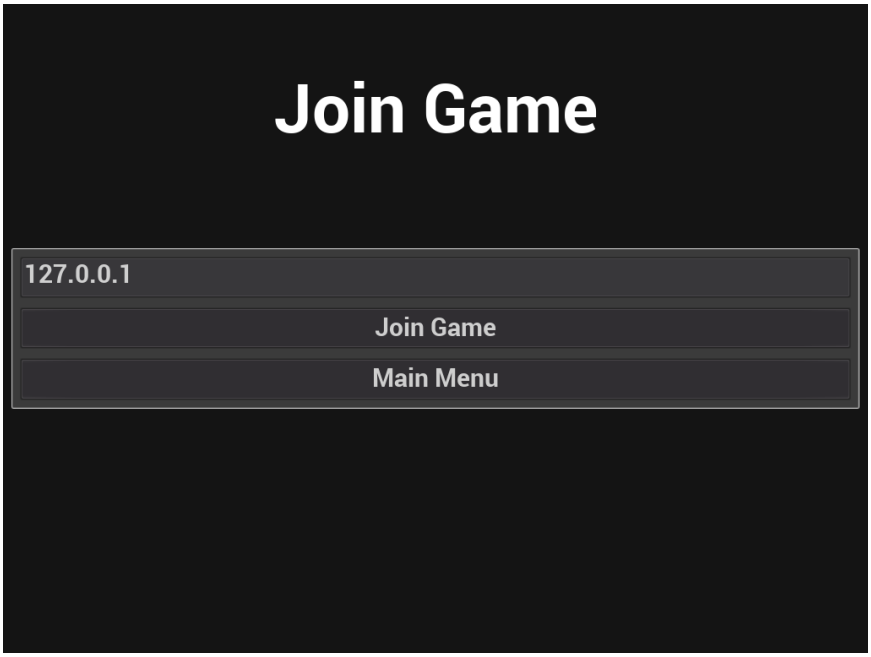

Figure 25: Join game menu design.

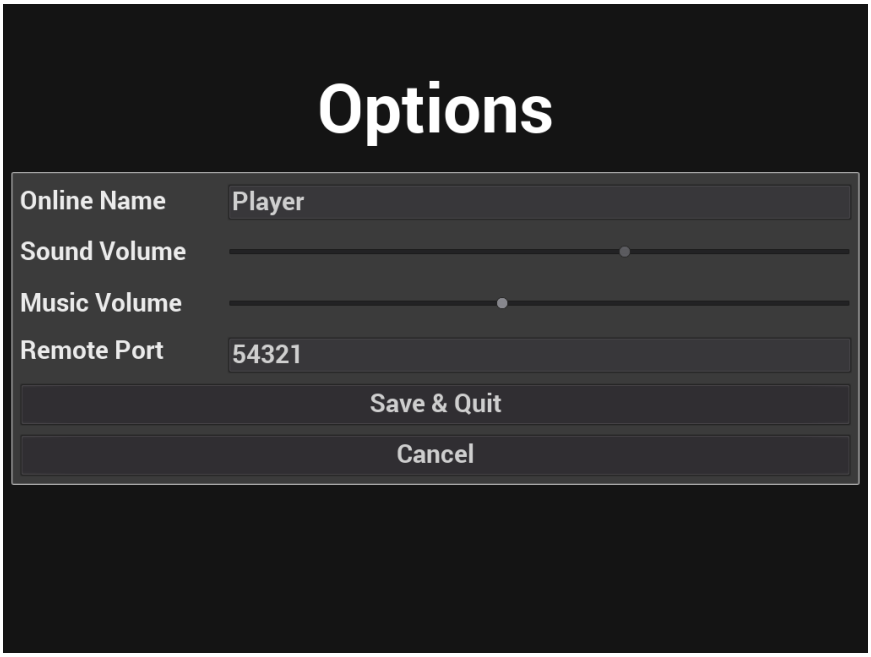

Figure 26: Options menu design.

7 System/Data Dependencies & Requirements

In order for a user to play Anttris, the user will need a personal computer with a modern graphics card, a mouse, and keyboard. The computer will need to be a standard x86 machine—running Windows, Mac, or GNU/Linux—that is up to date in order to play the game. The graphics card needs to be reasonably modern since it must provide a minimum of 30 frames per second. Most recent desktop and mobile cards should be able to achieve this. The mouse and keyboard are the primary input devices.

A user may also choose to play Anttris on a mobile device, in which the device is a smart phone, running either the Android or iOS operating system. The phone's operating system will need to be up to date in order to play the game effectively. The touch screen of the device will be used to input commands into the game.

In order for the user to play multiplayer matches, the user will need an internet connection and IP address for their personal computer or mobile phone in order to connect to the other user. The user can still play the game in single-player mode which does not require internet access, however game play will be limited.

Anttris will need filesystem access in order to create, write, and modify save data and scores for each puzzle. Additionally, the Puzzle Editor will need to be able to read and write data for user-generated puzzles.

8 Testing Plan & Results

In order to test that the game was working properly, we used Black Box and White Box testing methods. Using these methods we were able to test the game from the outside-in by making sure that the functionality of the interface preforms the correct actions then worked out way to the coding level of the game. This approach helped prevent a lot of errors and bugs during the early stages of testing, since we were not working with the code directly at the start. Any errors and bugs we encountered were documented and recorded for future reference.

Our testing plan is as follows:

1. Black Box Testing
2. White Box Testing
3. User Testing

For the first part of the testing plan, we tested the game on multiple physical computers to make sure the interface of the game preformed correctly. Testing the game on multiple machines allowed for a diverse testing experience, and confirmed whether or not our game can work on multiple systems. Once we knew that our

game could run on each of the tested machines, we began testing the interface of the game.

The interface of the game is what the user will be interacting with, so we wanted to make sure that the interactions of each function worked correctly and displayed the correct information to the user.

To test the interface of the game we:

1. Test the menu options
2. Test single-player Game Mode (Single-player, New Game, Continue Game)
3. Test Puzzle Editor/Creator: (Create, Edit, Delete)
4. Solve basic test puzzles (Test Puzzle 1, 2, 3. . .)
5. Multiplayer Game Mode

After testing the interface of the game and making sure each of the game options and modes work correctly, we will review errors and bugs we encountered during the Black Box testing. Upon completion we will need to begin White Box testing and testing that the correct values are being passed in the code.

8.1 Testing Menu Options

While testing the menu options, we clicked each of the available options to make sure that the selected menu option displayed the next options or pages the user needed to see. For instance, when we selected the single-player game option, it returned with the correct corresponding options. We tested each of the options the user may choose from and tested whether or not the user could reselect options by returning to the previous options. Testing this first was the most important since it will be the first thing the user will see when playing our game as well as deciding which type of game the user will be playing. Each of the options worked correctly and displayed the correct information needed.

8.2 Test Single-Player Mode

After testing each of the menu options, we moved on to testing the single-player Game Mode. In single-player, the user is able to select whether they wish to start a new game or continue a game they left off on. The New Game option allows the user to select the puzzle they wish to play and solve that puzzle from the beginning. To test this option, we set five test puzzles that were generated and tested each of the puzzles separately on each machine. We made sure that the puzzles generated

the correct puzzle and generated it the same way each time. After we tested the new game function, we tested the continue function. This function allows the user to continue from a saved game file and will allows the user to continue solving the puzzle.

8.3 Test Puzzle Editor/Creator

After testing the New Game and Continue functions, we tested the puzzle editor functions which are accessed in the single-player Mode. The puzzle editor is where the user can create new puzzles, edit previously made puzzles, and delete puzzles they may not want anymore. To test creating new puzzles, we created basic puzzles, seeing if we were allowed to place blocks on the grid correctly, and if the game would catch any errors while saving the puzzle. When Creating basic puzzles we used the in game tools to create and remove blocks from the grid. We tested to see if we could place each type of block, and if we could remove them after being placed on the grid. We used a similar testing method for the edit puzzle option since it is similar to the create new puzzle option. However we had to make sure that the puzzle the user wanted to edit appeared correctly in order for the proper changes to be made.

8.4 Solving Puzzles

After we tested the Puzzle Editor, we tested whether or not we could solve puzzles. Using pre-constructed basic puzzles, we solved each of the puzzles and confirmed that each of the blocks preformed the correct action when clicked, and tested if the game recorded the scores correctly and ended the game once the goal block was reached. Each of the tester puzzles, consisted of each type of block and possible interactions so we were able to make sure that the game flow was not ruined when blocks preformed their actions. Solving Puzzles was the last step in testing the single-player mode and was the most important to have working. In the end, the single-player mode worked how we wished.

8.5 Multiplayer Game Mode

The Multiplayer Game Mode was the last thing we tested since it involved the interactions between two computers. We tested the Create and Host Match options like we did with the menu options and the different game modes similarly to the single-player and Multiplayer game modes. The most important thing we needed to test for the multiplayer was the interactions between the two players. In order to test the interactions, we needed to make sure that the two players were able to connect to each other and preform actions in the interface and when solving puzzles with

each other. In the end, we got multiplayer to work between two players on the same network.

8.6 Unit Testing

Part of ensuring that critical calls in our code worked correctly was implementing unit testing. Fortunately, a member of Godot's community authored a handy test driven development framework, known as GUT, the [G]odot [U]nit [T]esting framework. [1]

Since we had decided early on to incorporate continuous integration into our workflow, we wanted to have our unit tests be run on Travis, so that our tests would run every time a build triggered. This alerts to issues immediately so nothing gets broken. We mainly tested methods used frequently to get and set various things in order to ensure that they were behaving as expected.

Hugo devised a Python script to interface with the Godot headless server on the test server so that the multitude of return values used were caught in a useful manner.

8.7 White Box Testing

The second part of the testing plan, we used the White Box testing method in which we will be dealing with the actual code of the game. We need to make sure that the code of the game is working correctly. We needed to make sure the code of the game was working correctly. Changes to the code was based on the errors and bugs we found during the black box testing. We treated the code as units and fixed only the function or object that needed fixing then reintegrated it back into the code. After reintegrating the newly fixed code, we needed to test the section of the game that we made changes to. By making the changes and fixing the problems with the documented bugs and errors, and breaking the code into units, we were able to efficiently make the necessary changes to the game. This was primarily done with the Puzzle Editor and the Networking between players. Once we made a group of changes, we tested the game again using the black box testing method in order to make sure the changes we made to the game did not affect any of the other sections of the game. Errors and bugs we found were documented and recorded and any necessary changes that were made. These changes were tested again on the physical machine. After these changes were implemented and tested effectively, we moved on to the next stage of testing; user testing.

8.8 User Testing

The last part of our testing plan is user testing. User testing allowed us to test the game as users and provide feedback about the game. This method of testing helped us gather information and discover any bugs or errors that we could not find during

the earlier parts of testing. For this part, we solved various preconstructed puzzles and gave feedback about the game. Using the feedback, we were able to make the necessary changes to ensure the game was fully functional and working properly before we presented the game.

9 Project Status and Summary

9.1 Project Status

Anttris began with many features in mind. Through the course of the development of Anttris, these features remained our central focus. Elements like competitive multiplayer and the puzzle editor were required to give the game the substance needed to make it a great game instead of just some puzzle game.

All of the main features we planned for the game to have were successfully implemented. Features like single player and the editor have been created and thoroughly tested. Multiplayer games can be hosted and played with friends. Players can save and load blocks. Every feature planned out in the beggining was successfully added and tested in the game.

In addition, Anttris has an official game website! The website features explanations of the game rules, convenient download links, and screenshots of the game. The website was developed by us, and can be viewed by pointing a browser at our github.io page.

Finally, Anttris was built using git and GitHub. This means that the whole Anttris development process and final code is available for download at our Github repository. The above two should be clickable links if viewed in most modern PDF readers.

In case they are not, here are the raw URLs: `http://gamewizards.github.io` and `http://github.com/gamewizards/anttris/`

9.2 Difficulties Encountered

Throughout the project's development, we experienced a few minor difficulties. Although we experienced difficulties, our design principles allowed for us to dynamically change our design and quickly integrate new ideas to fix old ones.

The biggest issue we ran into involved the game's original rules. While the rules were a neat idea, determining the solvability of a puzzle proved to be an NP-hard problem. After realizing the complexity of solving puzzles, we simplified the rules in a way that was interesting, fun and had a realistic solution-check time.

Another difficulty we had involved the Godot Game Engine used to implement Anttris. The software is still currently in Beta and as such, there are still minor issues throughout the application. One especially annoying Godot issue was version

controlling some of the resource files that were compiled into binary. This meant that only one person could work on scene files at a time. This lead to occasional conflicts when two people mistakenly worked on the same scene file. Another one was with using globals within Anttris. Due to the way that the scene tree works, any globally loaded objects in the scene tree get deleted on scene change. Most of the other issues were solved through digging into Godot's source code or by finding workarounds.

9.3 Journal of Project Activities

Team organization was important, so we organized weekly in-person meetings at various locations on campus. Most commonly we used Cramer 213, but also met in the library a few times. For all other times, we had all team communications centralized to Slack, so that everyone could talk to the team in an organized manner at any time of the day, as well as get centralized notifications from Github and Travis CI.

We used a modified Scrum methodology to organize the project. Agile development is great, but the normal schedules used by Agile teams are not necessarily optimal for students, so we settled on mainly using sprints as general guidelines for milestones. We also did not use daily in person standups, preferring instead to utilize Slack to talk constantly about what we were doing. Additionally, we combined retrospectives into our real time workflow, rather than having a meeting specifically for a retrospective.

9.3.1 Project Statistics

Below are some figures taken from both Slack (communication statistics) and Github (coding statistics.) Please note that Github unfortunately only reports statistics for the default branch (in our case, master.) Thus, this data may not accurately reflect work put into branches. This is particularly true of Figure 28, as it reflects a lot of the code that was merged into master towards the end of the project cycle. Additionally, Figure 29 is the state of the network graph on April 28 at about 10 PM. It will probably change a bit as more work is done, but it reflects how our branches have been fitting together.

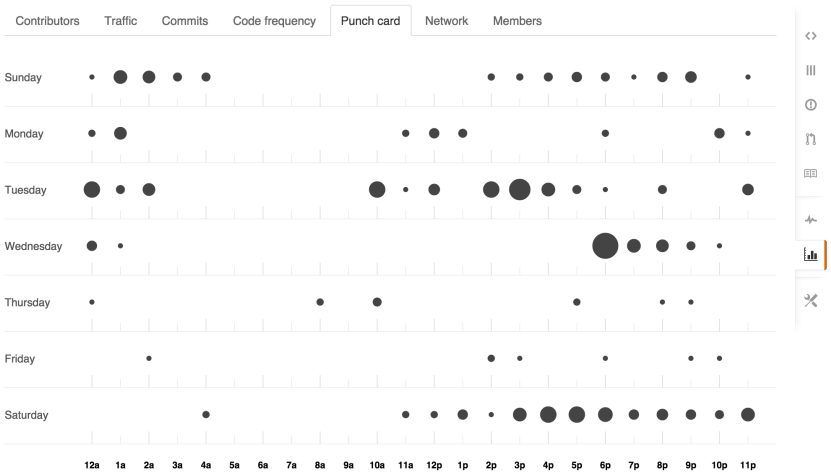

Figure 27: Github Punchcard

Figure 28: Github Code Frequency.

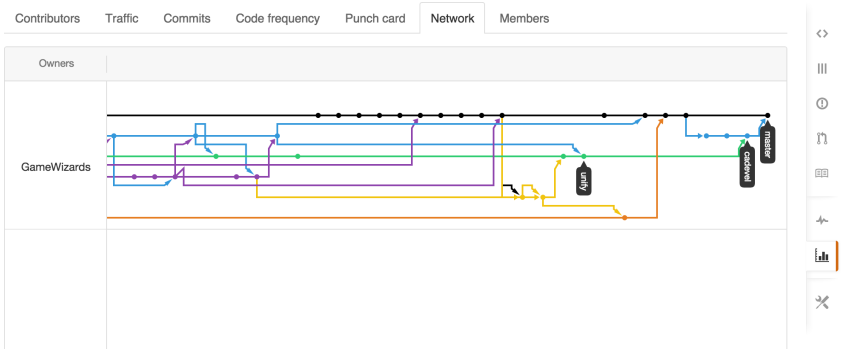

Figure 29: Github Network Graph Tail.

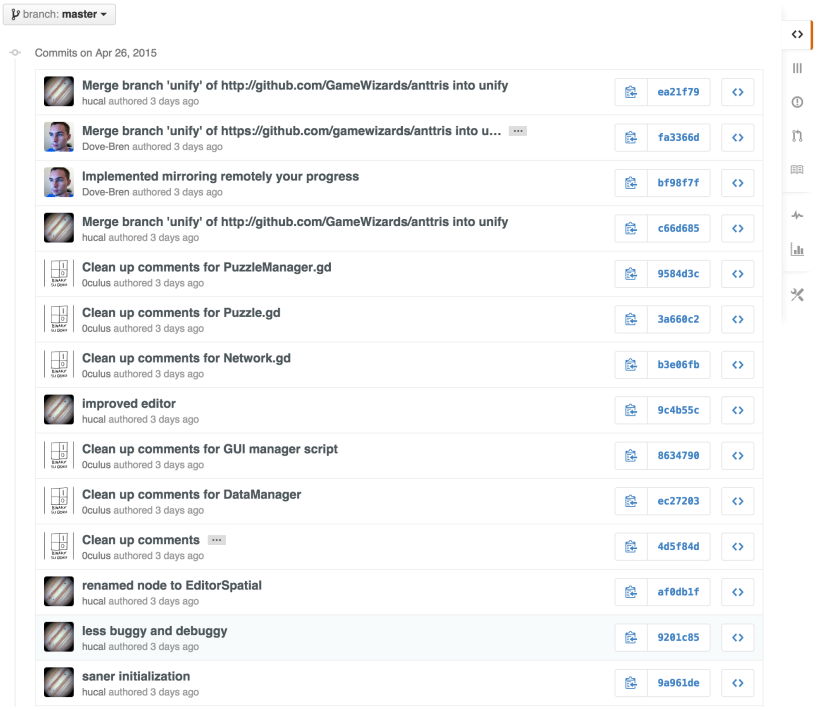

Figure 30: An Example of Commits to master.

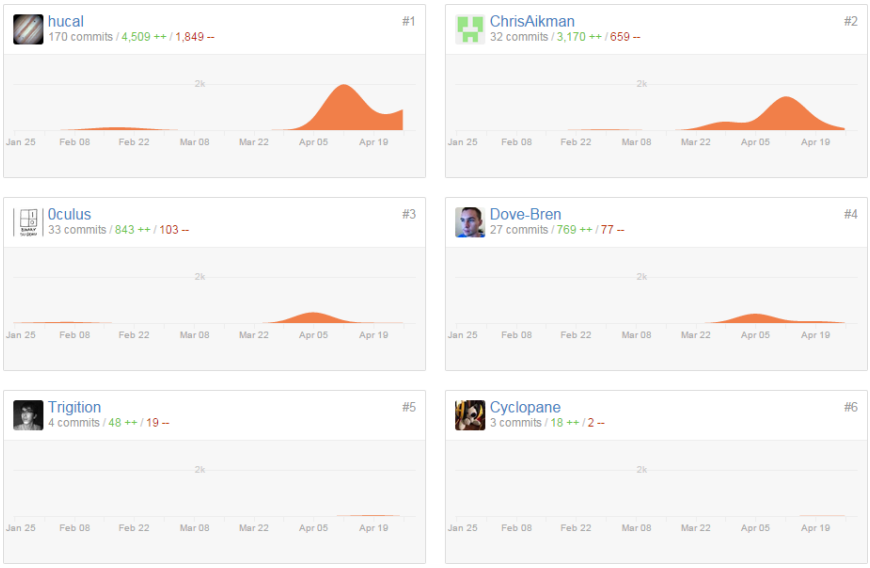

Figure 31: Total commits sorted by additions. 'Trigition' is Skyler Manzanarez pushing from a friend's computer.

As for Slack statistics, we are approaching 10,000 messages sent in total. We have multiple integrations, including Travis CI, Github, and Google Docs, as shown in Figure 32. Figure 33 shows an example of our Slack chat.

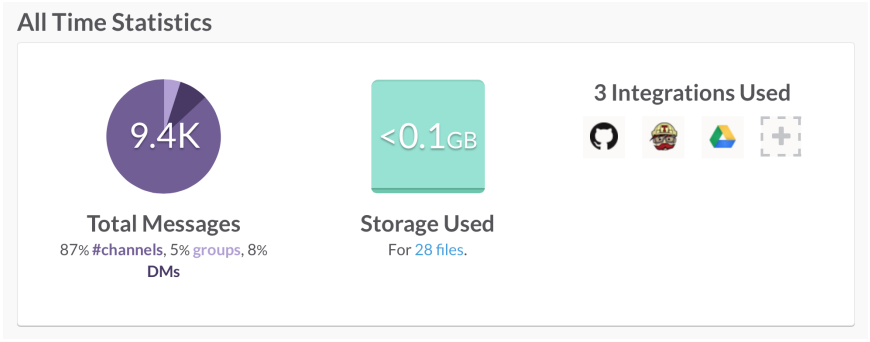

Figure 32: Our Global Statistics as Reported by Slack.

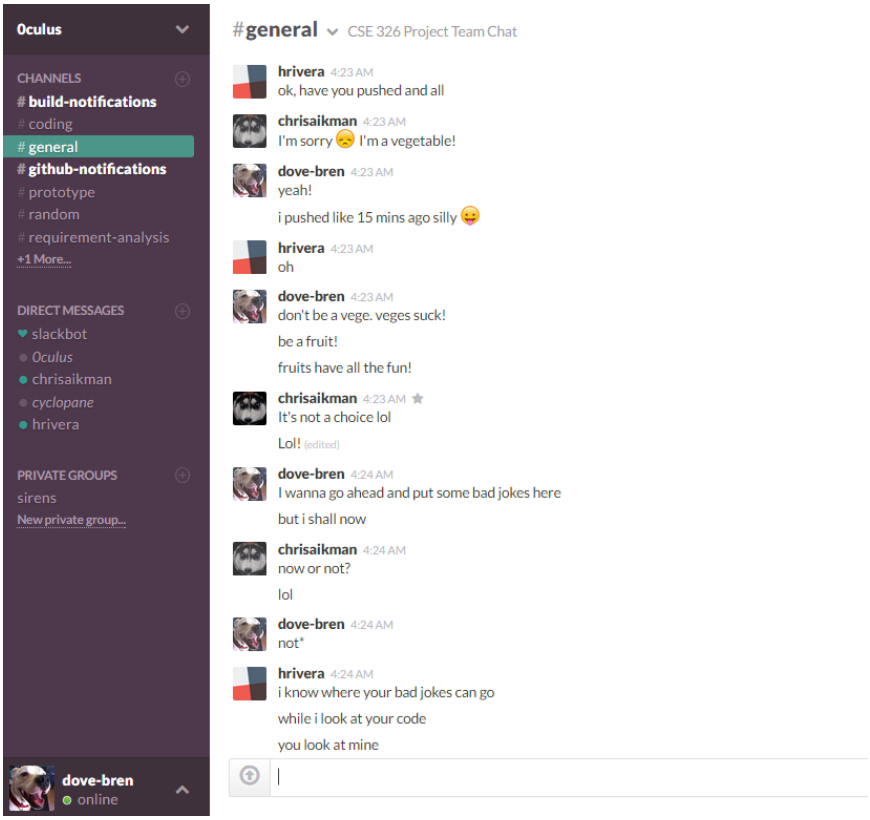

Figure 33: Our Slack.

9.4 Project Schedule

Here is our updated Gantt chart showing our progress.

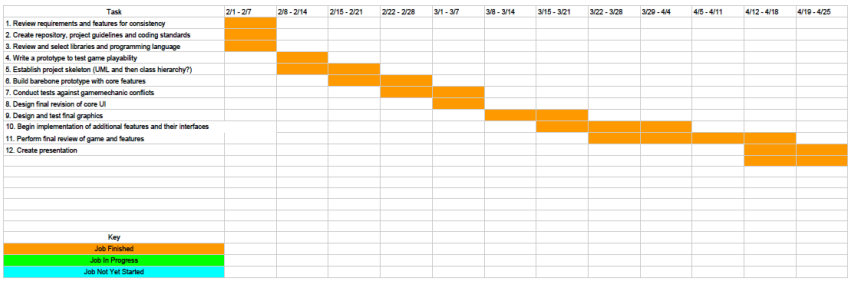

Figure 34: Gantt Chart for Project.

10 Appendices

Our project has a website at github.io (clickable link.) Our project repository is located on Github (also should be a clickable link.) In case those hyperlinks don't work, find the raw URLs below.

Website: `http://gamewizards.github.io`
Github repo: `http://github.com/gamewizards/anttris/`

References

[1] Butch Wesley. Godot Unit Testing Framework. `https://bitbucket.org/bitwes/gut/overview`, 2015.

[2] Juan Linietsky, Ariel Manzur, and Okam Studio. Godot Game Engine. `https://github.com/okamstudio/godot`, 2014.