Galactic Conquests

Dream. Build. Dominate.

Design and Implementation

Game Focus

The main decision was that Galactic Conquests would be more of a strategy game with real-time fighting components. Thereby forcing the player to focus on their overall gameplay strategy such as what route to take in order to get to a system (avoiding systems controlled by factions with negative standing), how to spend scarce galactic credits (taking a short-term focus buying missiles to win the next fight versus taking a longer-term focus on upgrading the ship) and diplomatic objectives (which faction to align with by accepting their missions).

The thinking behind this decision, was that it means players are likely to want to play for over 45 minutes playing the game instead of 10 minutes then get bored after a few space battles. Thereby giving the game the potential to make it big on steam like other similar games such as Faster Than Light (FTL) which puts a focus on the player developing a meta-strategy rather than individual battles, resulting in mild addiction and relatively fast spread of popularity of the game.

Game Architecture and Menu System

In order for the player to focus more on the meta-strategy, the decision made was to focus on developing a very elaborate and high quality menu system for the game. There were 16 different types of screens developed for the game. The code for these screens can be found in the 'cs4303.p2.game.screen' package. Two abstract superclasses for the screens were developed (*Screen* class and *AbstractSystemScreen,* a *Screen* subclass). Screen contains all the methods which needed to be implemented by a screen (e.g. draw, mousePressed, keyPressed, keyReleased). Along with these, the screen class contains all the code for creating buttons, listening for mouse hover over the buttons and game transitions when the button is clicked. AbstractSystemScreen abstracts away many of the common features for screens shown to a player while engaging with a system. Subclasses of this abstract class include BribeScreen, FightWonScreen, LeadershipOfferScreen, MissionScreen and SystemScreen.

Graph Theory

In order that mission rewards would higher the further away the target system was from the player's current system, the ability of calculating the minimum number of jumps to get from system A to system B was needed. The galaxy's network of system was treated as a graph. The algorithm found the shortest path from A to B using a breadth-first search strategy and then cycled through it in order to calculate its length. (For implementation, see calculateMinimumNumberOfJumps() in Galaxy class.)

Fight Screen and Physics

The physics element of the game takes place during battles. All battles are real-time and are around the system's habitable planet of the system. The planet exerts a gravitational pull on all objects. This is more noticeable the closer you are to the planet as the strength of the force pulling the object towards the planet is inversely proportional to the distance to the planet.

Collision Detection has been implemented through the use of the abstract DrawableObject class. Objects present in the fight screen include the DrawableShip (an instance created from a ship object which can be rendered to the screen and interacts with physics engine), the planet and the non-allowed area of gameplay extend the DrawableObject class. Each DrawableObject is able to generate a number of points around the edge of the object (when the getCircumferencePts() method is called). The collision detector runs when the checkForCollision() method is called on the FightState object every frame. It checks whether any objects have circumference points contained within any other objects (through calling the containsPt() method on the DrawableObject object). If a collision is detected, an impulse is applied to the offending objects effectively simulating a perfectly elastic collision.

Development Prioritisation

The game design was very ambitious, including physical, networking, AI and security elements. All of which I plan to fully implement as I plan to develop the game to a releasable-standard for both Steam and for its own website (http://www.galacticconquests.com).

As suggested in my feedback from Tristan on the initial Game Design, I decided that the networked (and multiplayer) version of the game, along with more sophisticated missions and AI was a lower priority.

Another low priority feature which wasn't completed (but partially implemented for being able to undo lost fights), was the ability to convert the entire game state into json and save it to file. Whenever the player entered a new system for the first time, the game would be automatically saved. Then when reopening the game, the option of reloading the saved game is given.

Furthermore, a large effort has been made to make the game as object-orientated as possible in order to reduce friction for future development.