F454 Project

Giny Huynh - 8095

Townley Grammar School - 14109

CONTENTS

SECTION 1

DEFINITION

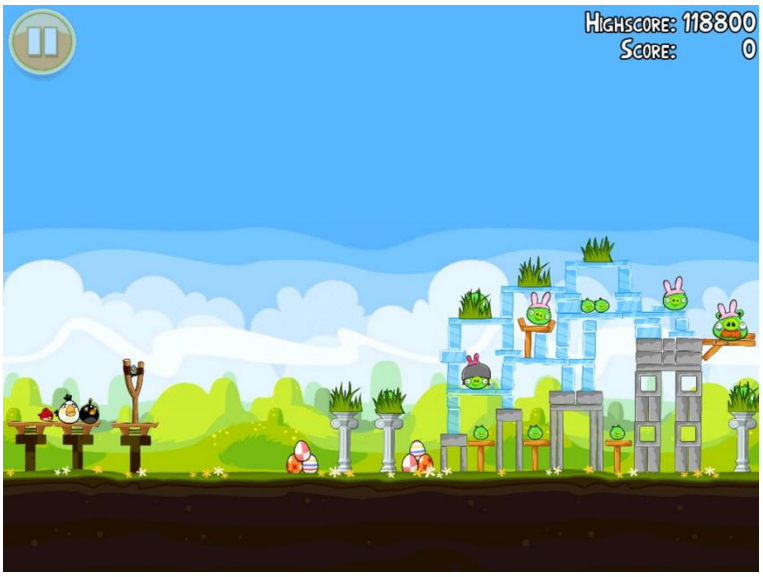
PROBLEM DEFINITION

Millions of people around the world regularly play computer games. There are a wide variety of games but there is a lack of games which reflect real-world physics. Since realistic games are true-to-life, even complex games are easier to understand and play, because the rules of the game are the same as how we think and see things in real life. Also, realistic games serve as great educational systems. This is why more realistic games are needed.

To address this, I will make a computer game that will be fun, competitive and most importantly, as realistic as possible, to give players knowledge that they will be able to use in everyday life.

CURRENT SYSTEMS

Here are some current games that have very lifelike physics simulations:



Angry Birds

This is a game where you catapult birds into the air and try and hit all the pigs.
The game uses projectile simulations, and the speed and angle at which the birds hit objects determines how they move. However, the projectiles are inaccurate as air resistance is not taken into account, and the birds' impacts will never be powerful enough to shatter/break the objects.
Image:http://www.gratisandroidapps.com/images/games/angry-birds/gallery/angry-birds-easter-screenshot.jpg

Call of Duty

These are a series of games which follow soldiers into war. Different editions of the game are set at different time periods: World War 2, the modern day, and 2025. But every game in the franchise is a first or third person shooter game, with a story mode or multiplayer mode. In the story mode, a single player completes missions to go along with the story and complete the game. Multiplayer games are over the internet.
The game uses extremely lifelike simulations such as in the movement of players and projectiles of bullets (and knives and grenades), where they bend slightly with longer distance shots. All the physics in the game is very accurate, but the games are unsuitable for children or families (18 certificates) as they are full of violence, drugs and foul language.
Image: (Call of Duty: Modern Warfare 4) http://www.theserif.net/wp-content/uploads/2014/03/Call-of-Duty-4-Modern-Warfare-Wallpaper-126.jpg

END USERS

The end user for this game will be anybody who will play the game. However, my target end user will be teenagers, as I feel that they will benefit the most from a lifelike game. Also, teenagers will have more ideas than adults about what physics simulations they want in a game, based on what they're learning at school. I will have a group of year 10 students (14 to 15 year olds) who regularly play games as my end users, as this is the average teenager age that this game will be aimed at. To find out what they want in the final product, I will arrange a number of interviews with them.

End users: Lyndon Huynh, Alex Tyler, Nakhil Chana, Daniel Voung, Oliver Hassan, Noah Robinson.

INVESTIGATION AND ANALYSIS

PROJECT PLAN

Project Plan	08/09/2014	15/09/2014	22/09/2014	29/09/2014	06/10/2014	13/10/2014	20/10/2014	27/10/2014	03/11/2014	10/11/2014	17/11/2014	24/11/2014	01/12/2014	08/12/2014	15/12/2014	22/12/2014	29/12/2014	05/01/2015	12/01/2015	19/01/2015	26/01/2015	02/02/2015	09/02/2015
Definition	Completed																						
Investigation and Analysis																							
Nature of the Solution																							
Algorithms																							
Test Strategy																							
Software Development															Christmas								
Testing															Break								
Documentation																							
Evaluation																							

INITIAL INTERVIEW AIMS AND PLAN

The purpose of the first interview is to get a more refined idea about what my end user wants for a game. Since there are six end users, the interview will be in the form of a questionnaire. I have chosen to use a questionnaire as it ensures that all users answer the same questions, the answers come quickly and that they are formatted in a way where it is clear which responses are the most popular. The most popular responses will shape the basic structure of my game.

I will give this questionnaire to my six end users:

The last question is in the questionnaire as I will not be speaking face-to-face with my end users. Therefore, if they have any other comments that would have been said in an interview, they can write them here.

QUESTIONNAIRE

1) Which of these types of computer games do you enjoy playing the most?

☐ Arcade ☐ Strategy ☐ Puzzle ☐ Shooter ☐ Other (please specify) _____

2) Do you prefer games that have levels (e.g. Angry Birds) or continuous games (e.g. Flappy Bird)?

☐ Games with levels ☐ Continuous games

3) What physically realistic games have you played recently?

_____
_____
_____
_____

4) What aspect/s of these games do you dislike (if any)?

_____
_____
_____
_____

5) Which of these physics simulations would you like to see in the new game (tick all that apply)?

☐ Radioactive decay ☐ Balancing ☐ Buoyancy ☐ Electricity ☐ Bending light
☐ Magnetism ☐ Pressure ☐ Motion ☐ Projectiles ☐ Waves
☐ Other (please specify): _____

6) Are there any specific features that you would like to see in the new game?

_____
_____
_____
_____

INITIAL QUESTIONNAIRE RESULTS

These are the results of the questionnaires. Each end user's responses are written out in full.

	Lyndon	Alex	Nakhil	Daniel	Oliver	Noah
Q1	Arcade	Arcade	Shooter	Strategy	Arcade	Strategy
Q2	Games with levels	Games with levels	Continuous games	Games with levels	Games with levels	Continuous games
Q3	Angry Birds, Pocket Tanks, Fruit Ninja	Call of Duty: Ghosts, Osmos	Call of Duty: Ghosts	Pocket Tanks, Pool Master	Angry Birds, Osmos, Pool Master	Pocket Tanks
Q4	Unlocking items with real money (Fruit Ninja) Can't see whole map when catapulting the birds (Angry Birds)	None	The game is expensive	None	None	Game is slow with 2 players, but too difficult when against the computer
Q5	Bending light, pressure, motion, projectiles	Radioactive decay, bending light, motion, projectiles	Balancing, pressure, motion, projectiles	Bending light, motion, projectiles	Bending light, motion, projectiles	Electricity, motion, projectiles
Q6	Save game automatically, a free game	Cool music, clear instructions on how to play game	A free game (to buy and in-game)	High scores	Ability to save game automatically	High scores so can face friends

None of the end users answered 'Other' for questions 1 and 5.

ANALYSIS OF INITIAL QUESTIONNAIRE

Upon looking at the table of results, it is I can see that:

- Arcade games are the most popular type of games
- Games with levels are preferred over continuous games
- The most popular physics games played at the moment are Pocket Tanks, Call of Duty: Ghosts, Osmos , Angry Birds and Pool Master
- The biggest problem in these games are the fact that some are not completely free

- All end users want motion and projectile physics in the new game
- Additional features desired in the new game are high scores, automatic saving and no in-game purchases (with the game being free too)

My game will be based on a physics simulation, to have an educational game which is realistic. Since all of my end users wanted motion and projectile physics, I have chosen to base my game around projectile physics. This is because projectile physics incorporates motion with gravity. I will now research what is needed in a projectile based game to make it enjoyable to play.

Evolution of projectile games

The most popular form of projectile games are artillery games, which have been around for a long time, and continue to evolve, each time trying to make the games more and more enjoyable.

Artillery games started out as two player games, with both players using a single (same) weapon, taking turns to shoot each other. These games would be fun to play, but only for one round. So, the games were improved by having more than two players on the screen. Also, there'd be new weapons where the players could choose which to use, and larger playfields. This made artillery games more enjoyable for a much longer time; however, it spiraled out of control when there were so many players on the playfield that you didn't need to aim accurately at all to be able to hit somebody. This made the game less strategic, and therefore less fun, but people still enjoyed playing with lots of people at a time.

So, the games were tweaked in that the playfield was extended by an enormous amount – several times bigger. This improved the enjoyment of playing artillery games greatly, as it meant that there were many players but there was still a strategic element to them. This was especially great for playing on larger screens, but with small ones, there was too much scrolling to find the other players. Nonetheless, artillery games developed more and more by having many more weapons, each having a different, wild style, but keeping the retro graphics. Fun sounds were also added to the game so that players didn't take the game too seriously and so ruin the fun of it.

From using this core structure for successful projectile games, many different styles of games were later developed. This includes shooting across planets to kill each other (e.g. Warheads), players ejecting themselves across the map to kill each other (e.g. Worms), or games which didn't include shooting or killing at all (e.g. Lunar Lander).

The game that I make must appeal to the majority of my end users, so now I need to research and analyse the games which they currently play. I will focus on Pocket Tanks and Osmos, as these games incorporate lots of motion and projectile physics.

Pocket Tanks

Pocket Tanks is a one or two player game where the aim is to apply as much damage to the opponent's tank as possible, and the one inflicting the most damage wins the game (there is no killing). The players take turns to attack each other – each turn consisting of using one weapon and moving up to four times. Each player has a range of weapons to choose from, the range being what they chose before the game

begins. Each weapon can only be used once, and with every weapon, the player chooses the angle and power rating, before pressing ‘Fire’ to see what happens. Each round the damage is added up as points. The player with the most points under their name after ten rounds wins the game.

This game can be multiplayer (maximum of two players) or against the computer. The computer has a range of difficulties (from 1 to 10), which the user selects before the weapon select. Players (or the computer) choose weapons in turns, one weapon at a time from one list. There is an option to randomise the weapon select to make the process quicker. There are initially 35 weapons to choose from, which come with the free game, but you need to pay to unlock more weapons. There are up to 120 weapons to get altogether, which means it would cost around £13 unlock the whole game.

Gameplay: http://igra.igra.pustyshek.net/uploads/posts/2010-12/1291927958_1291926413_pocket-tanks.jpg

Weapon select: http://www.blitwise.com/ptanks/pt_weaponshop.jpg

The graphics of Pocket Tanks are rather minimalistic, which gives it a retro look. There is no gore and no-one dies, making the game family friendly. And instead of blood or flying tanks parts, the damage (in numbers) floats off the hit tank. The playfield is a single screen’s size, but you can zoom in and out (although only a little bit of zooming is required). The player can choose the type of terrain to play on. The terrain can change during the game as there are weapons which drill through and attack the ground. The map also has winds which vary slightly each round to mimic winds in real life.

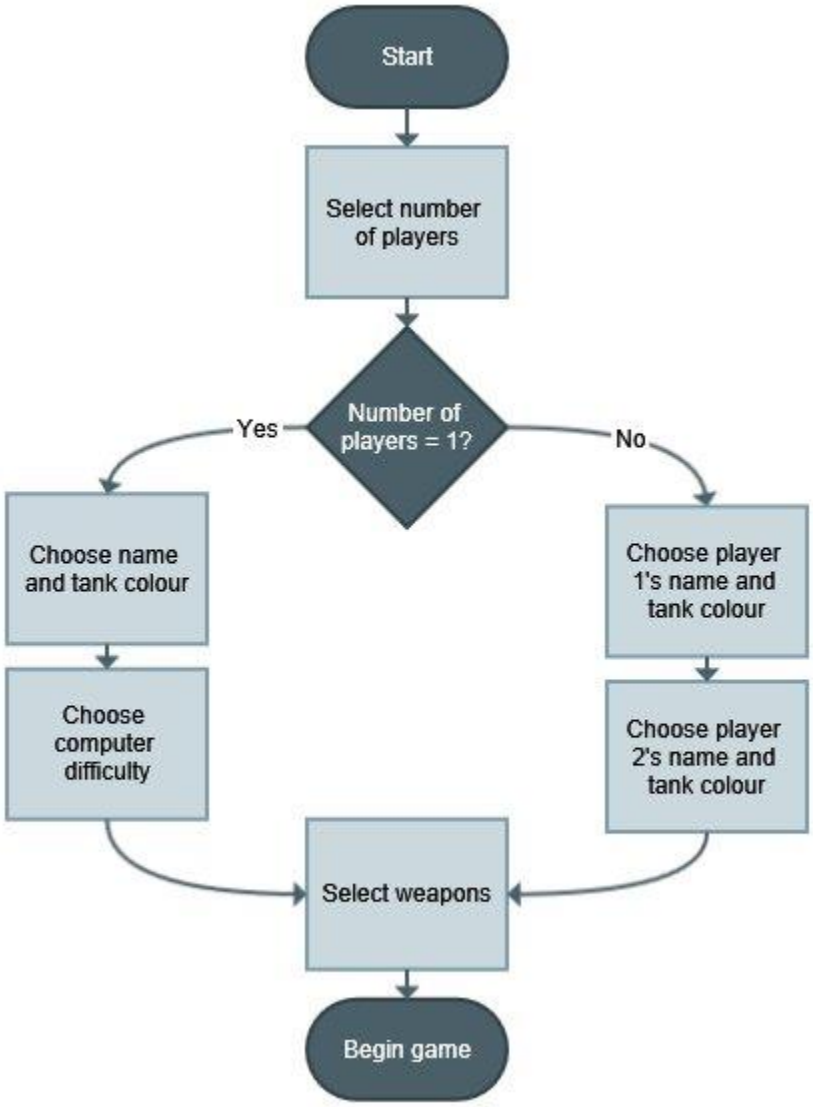
The player has other customisation options such as being able to disable some weapons (if some are too powerful or to change the style of play) and control sounds.

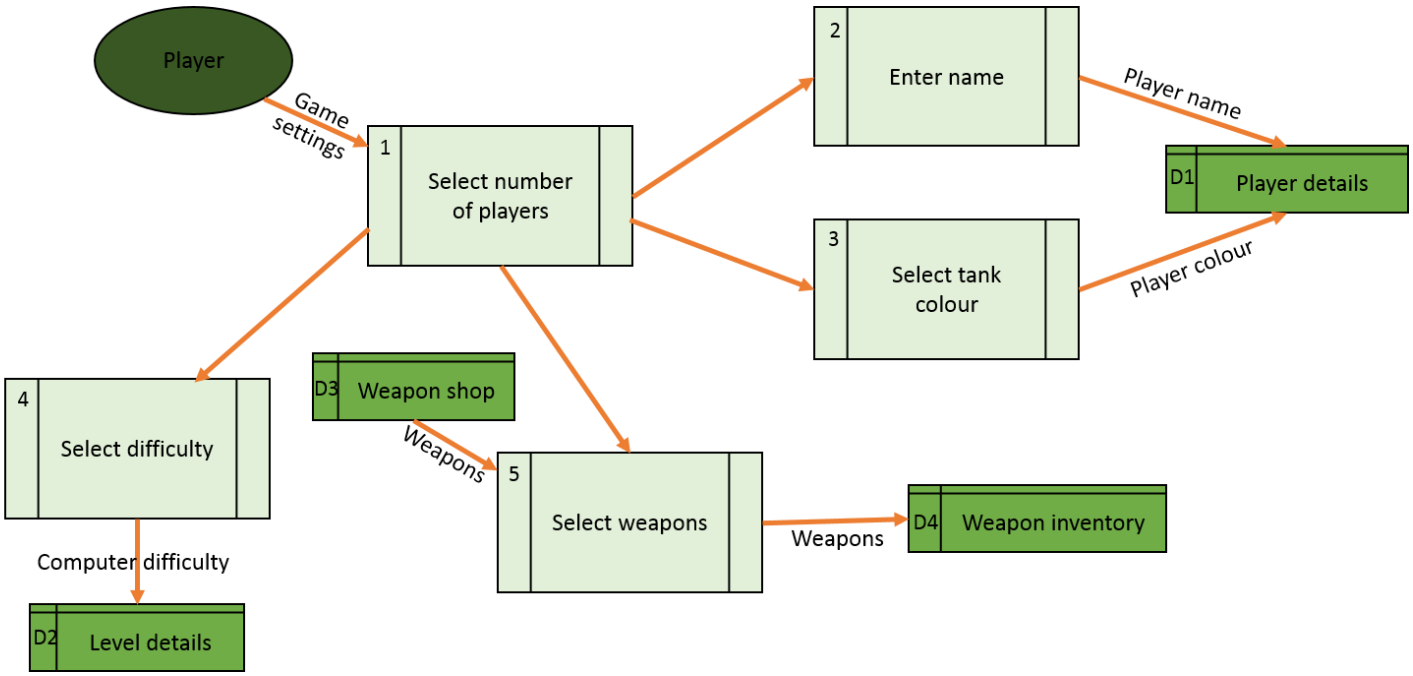
Many of the features in this game make it so popular, such as the different maps, the range of weapons and the graphics, but there of some features which I think could be improved. This includes the scoring system (killing the opponents tank would be more satisfying), and that the player can't select the angle or power by simply dragging the cursor to the desired angle or distance from the tank.

Changing terrain:
http://wscont1.apps.microsoft.com/winstore/1x/709608bb-b283-4927-9034-498639ba495b/Screenshot.74890.1000001.jpg

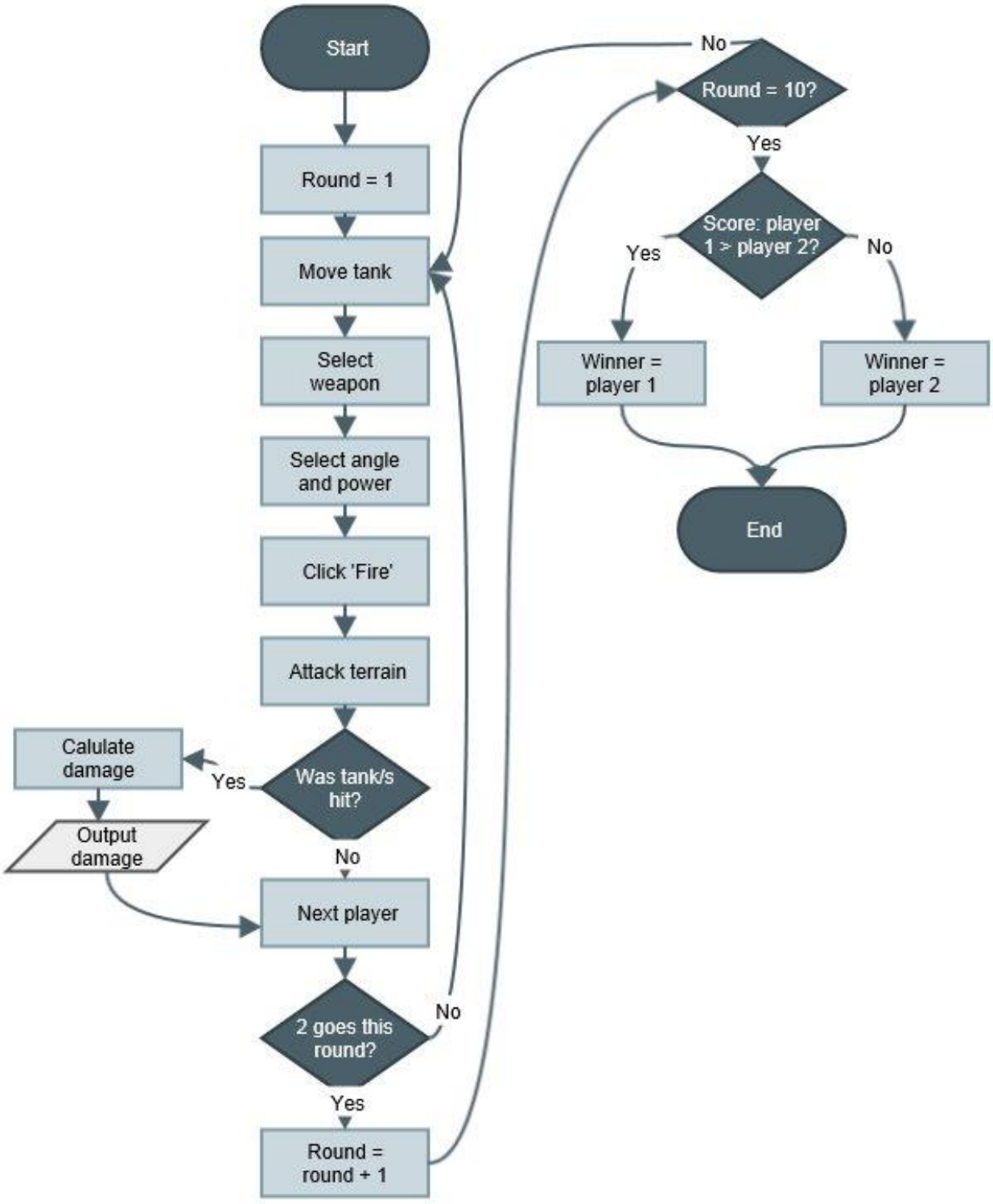
Pocket Tanks Flowcharts and Data Flow Diagrams

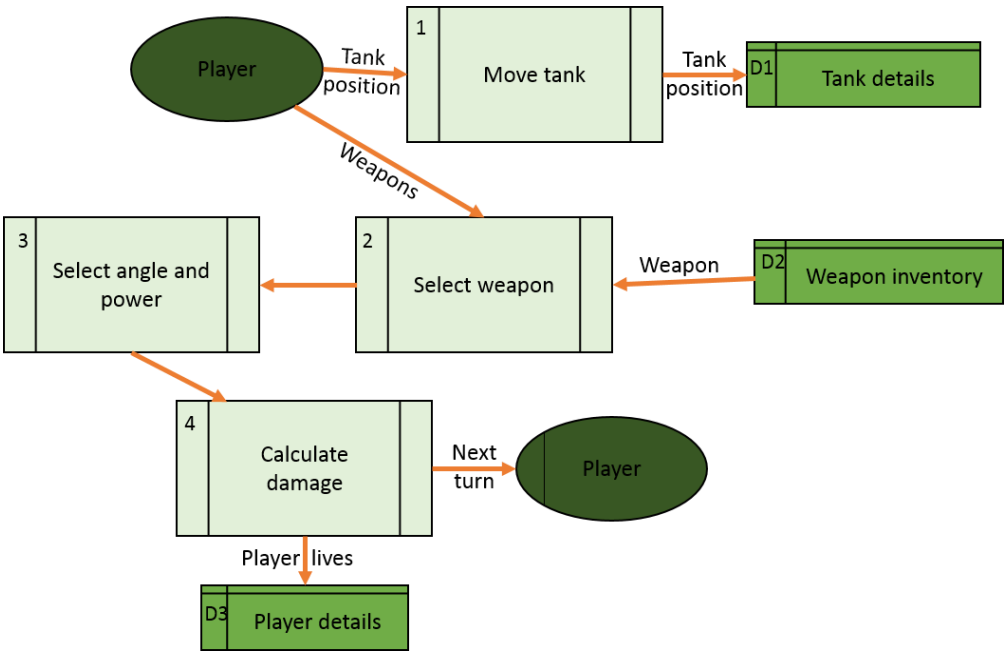
Game Select




Player
Game settings
1
Select number of players
2
Enter name
Player name
D1
Player details
3
Select tank colour
Player colour
4
Select difficulty
Computer difficulty
D2
Level details
D3
Weapon shop
Weapons
5
Select weapons
Weapons
D4
Weapon inventory

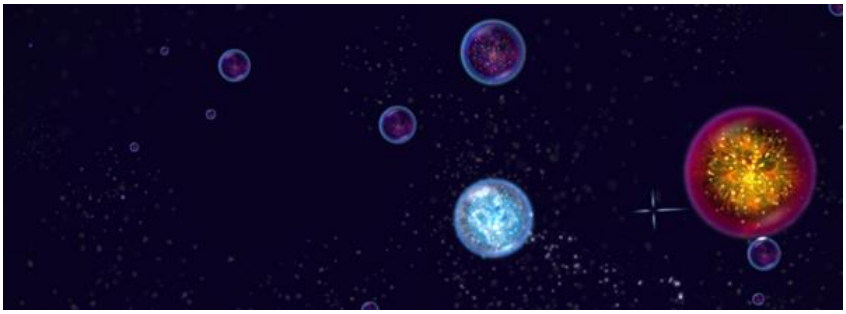
Gameplay


Start
Round = 1
Move tank
Select weapon
Select angle and power
Click 'Fire'
Attack terrain
Was tank/s hit?
Yes
Calulate damage
Output damage
No
Next player
2 goes this round?
No
Yes
Round = round + 1
Round = 10?
No
Yes
Score: player 1 > player 2?
Yes
No
Winner = player 1
Winner = player 2
End



The game uses uncomplicated physics such as parabolic projectile motion, realistic motion of liquids with gravity and different attack damages depending on weapon and angle of impact. It was written in C++, a language that is the norm for games as it can get much quicker running speeds than languages such as Visual Basic.

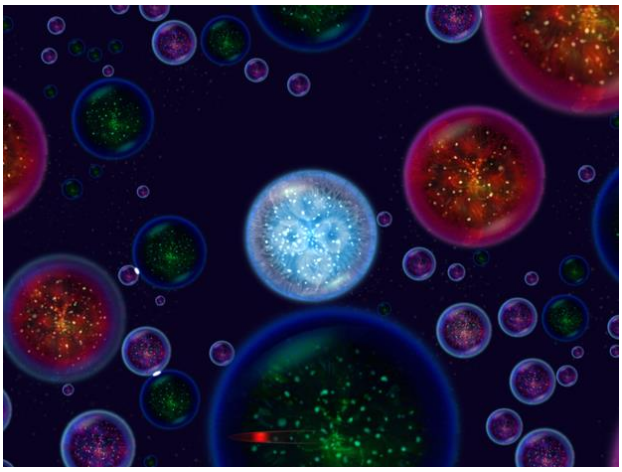
Osmos



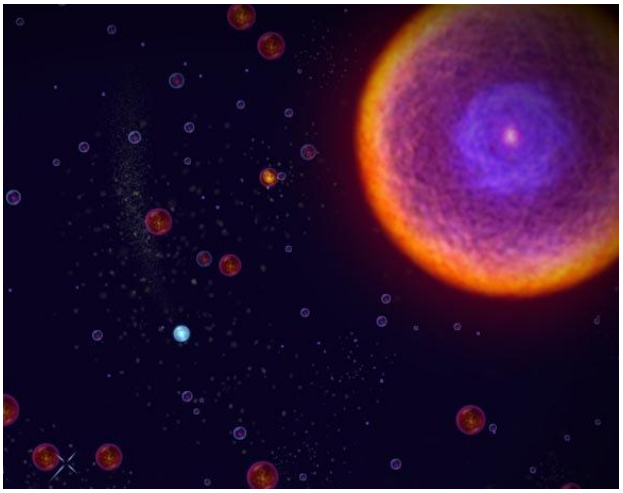
Gameplay: http://www.rockpapershotgun.com/images/09/aug/osmos1.jpg

Osmos is a one player game where the objective is for the player to "become the biggest" (There is a multiplayer mode, where the objective is to absorb the opponent). The player is a particle called a mote, and to get bigger, the mote must absorb other motes, but it can only absorb motes smaller than itself. This also means that the player is at risk of being absorbed by larger motes. The player absorbs motes by propelling itself towards the other motes, but in doing so, it must sacrifice some of its mass, and so gets smaller. The mote ejects its own mass to move – this uses motion and momentum physics (Newton's third law: for every action, there is an equal and opposite reaction).

There are different levels in the game, each level being a different type of game. There are some levels which use antimatter physics: there are anti-motes which, when collide with motes, both annihilate. There are also levels which use gravitational physics: there is an attractor at the centre of the map which has a larger gravitational pull, and motes orbit it (like the sun and planets in the solar system). There are other



Antimatter level: http://www.nonfictiongaming.com/wp-content/uploads/2012/02/osmos-14-35-59.png



Solar level: http://createdigitalmusic.com/files/2009/09/osmos2.jpg

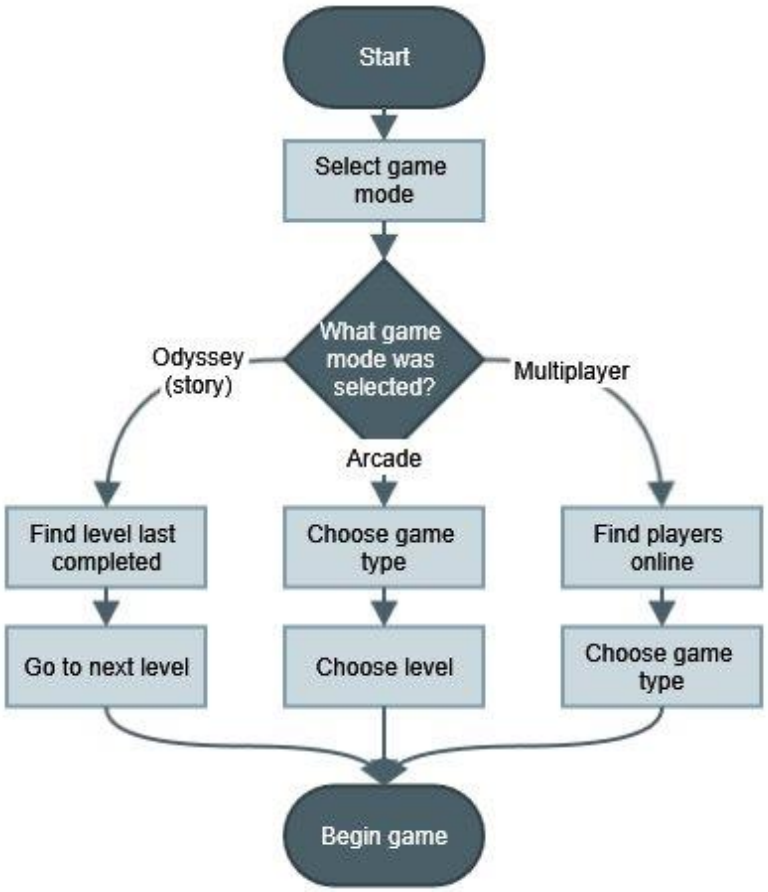
levels such as 'Repulsor' and 'Sentient' levels which feature new types of motes, and 'Warped Chaos' and 'Epicycles' levels, which use the same physics shown throughout the game to create more complex levels.

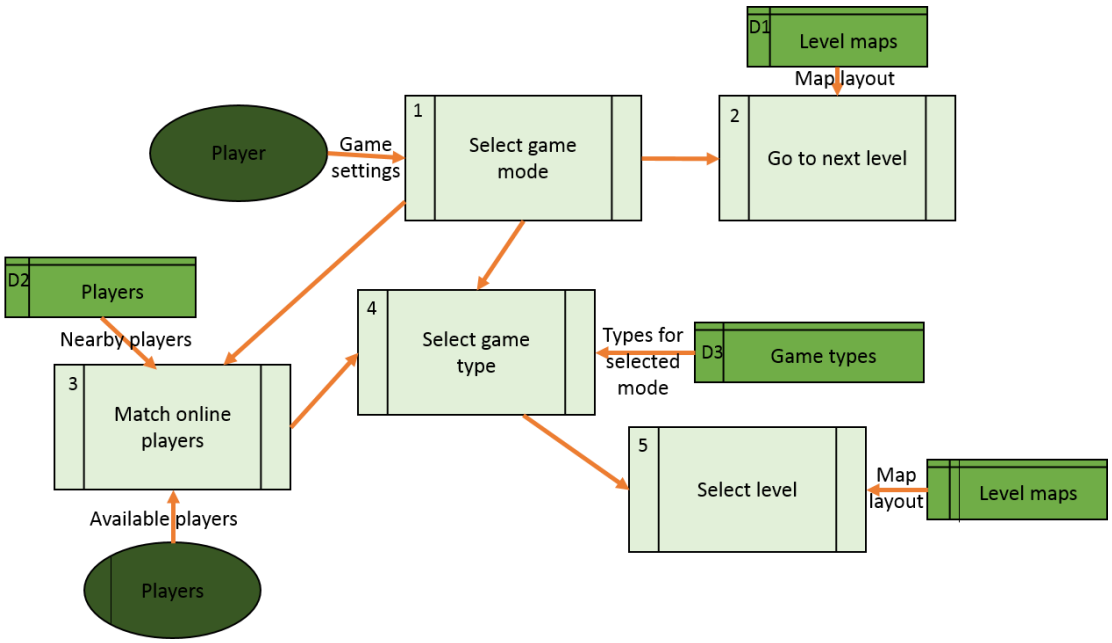
A big selling point of this game (which is available to buy on PC via Steam, or on iTunes store or Google Play store) is its calm atmosphere created by the crisp graphics and serene sounds. OpenGL was used for the graphics of Osmos as everything on the screen is translucent, making the game's graphics much more complicated to code. The music for Osmos is all very tranquil, making the game very relaxing to play. The simplicity of the music means that it can be slowed down or sped up without sounding distorted (the speed of the music changes depending on the speed of the game, which the player controls).

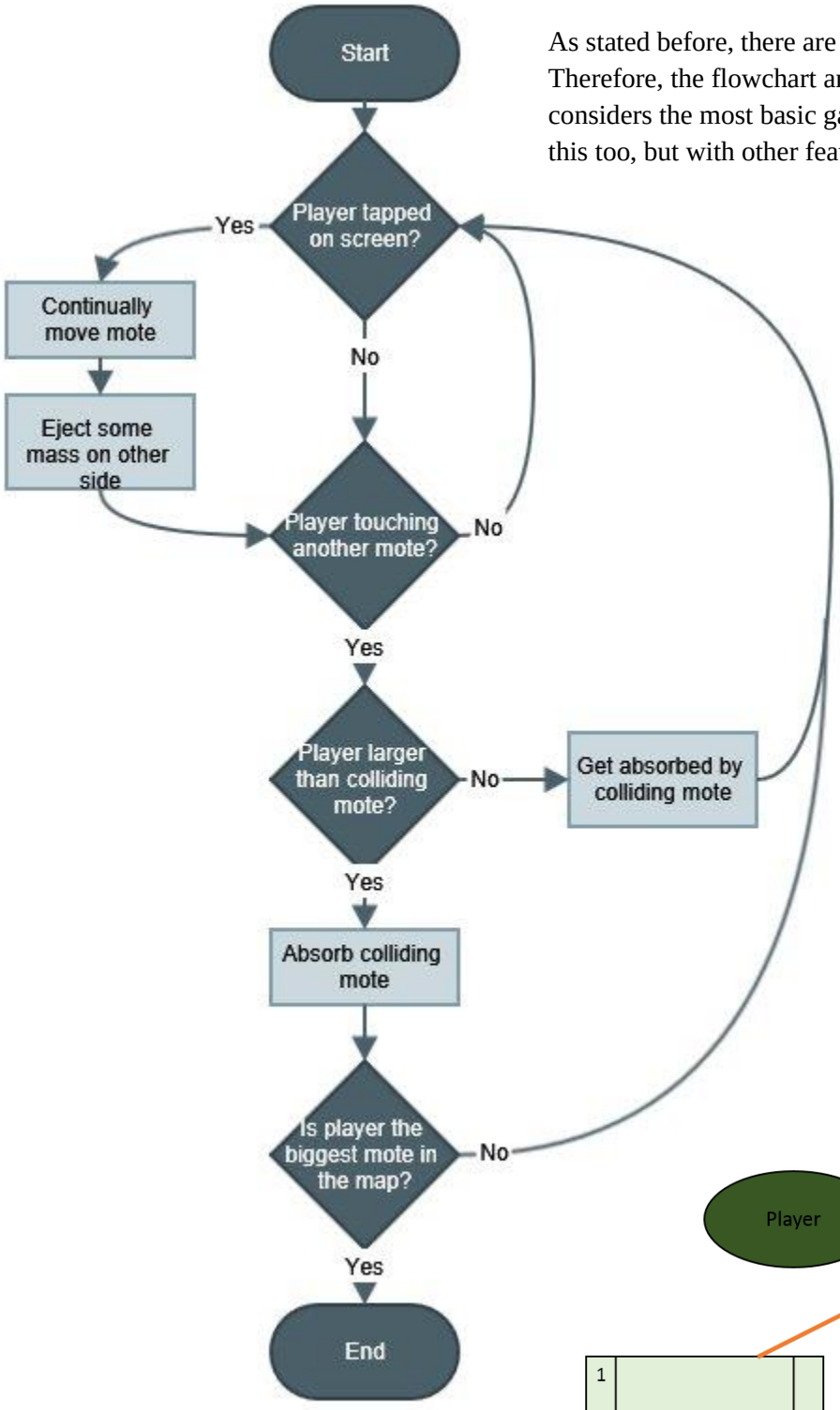
Every map is either rectangular or circular, with the same moving background. The player can zoom in and out, with the player's mote always at the centre of the screen.

Many aspects of Osmos make it such a unique game, however I find that the soothing music sometimes contradicts with the intensity of the more complex levels of the game.

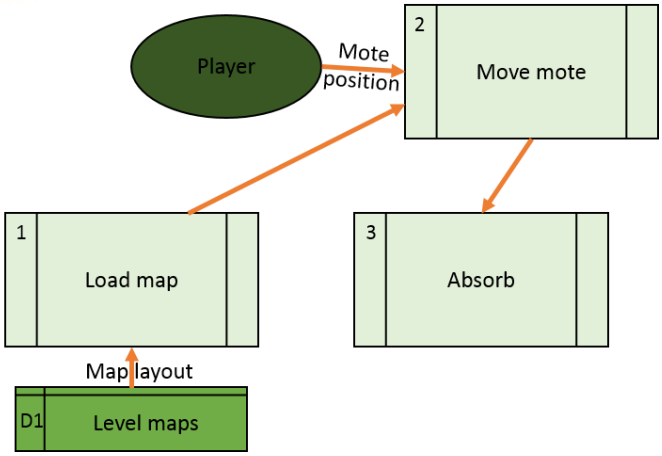
Game Select


Start
Select game mode
What game mode was selected?
Odyssey (story)
Arcade
Multiplayer
Find level last completed
Choose game type
Find players online
Go to next level
Choose level
Choose game type
Begin game


D1
Level maps
Map layout
Player
Game settings
1
Select game mode
2
Go to next level
D2
Players
Nearby players
4
Select game type
Types for selected mode
D3
Game types
3
Match online players
5
Select level
Map layout
Level maps
Available players
Players



As stated before, there are many level types in Osmos. Therefore, the flowchart and data flow diagram only considers the most basic game mode. All other modes use this too, but with other features developed into it.



Osmos was written in Objective-C, which lets programmers mix C and C++ modules. These languages are particularly beneficial to use when making the game across different platforms.

The user requirements of my game so far are:

- Must be an arcade game with levels
- It must be projectile based
- It must be free to play
- It must include high scores and the ability to save the game

The favourable features from Pocket Tanks and Osmos that I may include in the game are:

- Family friendliness (no violence/gore)
- Levels having a variety of difficulties
- Retro, colourful aesthetics
- Few controls (e.g. keyboard only game)
- Fitting music

FOLLOW-UP INTERVIEW PLAN

To incorporate as much of the above as possible, I have come up with a brief of my game*: launching things in outer space*.

To get more specific user requirements I have planned some questions for a follow-up interview. The questions that I will ask my end users are:

1. Are you happy with the requirements so far?
2. What will the aim of the game be?
3. What do you want the graphics of the game to be like?
4. What do you want the sounds of the game to be like?
5. How do you want each level to differ?
6. What will be the name of the game?

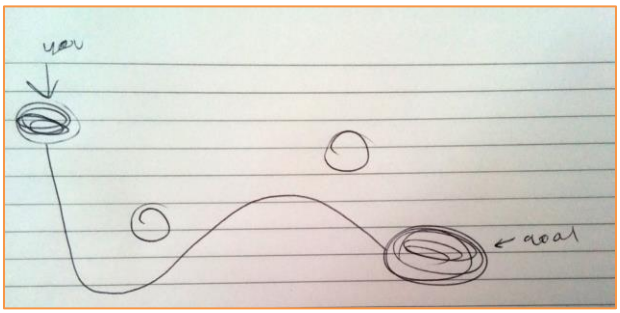
I will tell my end users the requirements of the game so far and the features of the games I analysed that I liked, before starting the interview. The interview will be with all of my end users together, so that there can be discussions about what they want in the game.

INTERVIEW TRANSCRIPT

Key: **Me** ■, Lyndon ■, Alex ■, Nakhil ■, Daniel ■, Oliver ■, Noah ■, All ■

1. **Are you happy with the requirements so far? Is there anything you disagree with?**
   - Everything is fine.

2. **What will the aim of the game be?**
   - Kill all the aliens.
   - Shooting from a spacecraft?

- No, that's a too generic game.
- What about shooting from a planet so that there are projectiles from the gravity?
- Ok, but you can't have blood and guts flying everywhere because you…
- **Yes, I'm going for a family friendly game.**
- Then instead of shooting aliens, you can shoot planets because they're inanimate. It can be like a war between planets.
- That's not really an arcade game though, is it? Especially because you want simple controls, right?
- **Yes, it should be reasonably simple, but I don't want it to be a matter of hammering the mouse without the player thinking about the physics behind the moves.**
- Then let the player control the 'bullets'.
- The 'bullets' should be actual rockets, and we control the angle and power at which they're fired.
- Like Pocket Tanks? Because that wouldn't be very fast paced.
- Then it'll be a chill game, where you can take your time to aim and fire.
- You could also do it like Osmos, where once you reach an objective, you go to the next level. And the objective could just be to fire a rocket to a certain place, like a destination planet.
- Just one rocket? That's not going to be very hard.
- Then you can put other planets in the way so that the path of the rocket isn't a simple curve. And so the player has to avoid crashing into the other planets.
- But you can also use the gravity of those planets to help you get to your destination, like:
- 

Oliver's sketch during interview to demonstrate his idea

- Ahhhh.
- **So are we all agreeing that the aim of the game is to launch a rocket to the destination of the planet and go to the next level?**
- Yes, and we control how the rocket is launched in the first place.

3. **What do you want the graphics of the game to be like? Cartoon planets and rockets or lifelike ones?**

- Cartoon.
- So that crashing into a planet looks like a bad thing, but crashing into *the* planet is a good thing.

4. **What do you want the sounds of the game to be like?**

- Fun, not serious.
- Fun being in music or sound effects?

- **Do you want background music in the game?**
- Yes.
- But something simple and repetitive so we don't focus too much on it.
- But it has to be fun and catchy.
- **And the sound effects?**
- Fun too. So for instance, when the rocket crashes, there's a "boooosh", even though we know a rocket can't make that much of an impact.

5. **How do you want each level to differ?**

- It gets more difficult.
- The target planet is different.
- It doesn't even need to be a planet – just something big.
- **What about the in-between planets?**
- They can be anything – doesn't matter.
- But in different levels they should have different masses to have different gravitational pulls.

6. **What will be the name of the game?**

- Something about landing a rocket.
- Rocket Lander.
- Sounds too similar to Lunar Lander.
- Something about controlling a rocket?
- Rocketeer?
- That's a superhero.
- Rocket Launch?
- Boring! But something about launching is good, because that is mainly what we're doing.
- …
- You know in films where there's a rocket launch? Well, the speaker man always says, "3…2…1…Launch!" That could be the name of the game.
- Or just after that, when he says, "And we have lift off."
- It's a cool phrase, but it doesn't work as a game title.
- What about just before the "3 2 1 launch" when the guy says, "All Systems Are Go"?
- I like that! Because you're – the player – is controlling the rocket, making everything ready for lift off.
- Silly question: What does "All Systems Are Go" actually mean?
- It means everything is finalised and the spacecraft is ready to launch.
- **Then 'All Systems Are Go' it is?**
- All Systems Are Go.

ANALYSIS OF INTERVIEW

Seeing as all of my end users were satisfied with the previous user requirements, I shall build upon that. Form this interview, I can see that:

- The game will be about launching a rocket to a destination planet
- The projectile motion will be caused by the gravitational pull of all the spatial bodies on the map
- The game won't have time constraints
- The graphics will be cartoon-like
- There will be background music as well as in-game sounds
- The music and sound effects will be comical
- Levels will have a range of difficulties and different destination planets
- The mass of the other spatial bodies will change each level
- The game will be called 'All Systems Are Go'

Programming Languages

Because what I will be making has more complex projectile physics than graphics, I researched what programming languages are most suitable for physics simulations. In a Stack Overflow discussion about "What's the best language for physics modeling?" I found out that:

- C++ is more suitable for very complex physics simulations
    - Advantages: it can handle more complex data structures and more quickly than other languages
    - Disadvantages: the code is very clumsy for smaller programs, as a lot of time is needed to write complicated loops
- Python is more suitable for small physics simulations
    - Advantages: it is much easier to write the physics of the code as it is the better language for data manipulation, and it's the most popular language for physics simulations
    - Disadvantages: it is slower than other languages because it is an interpreted language and doesn't work with objects as efficiently

I have chosen to write my program in Python because the simulation in the game is small-scale and there is much more online help with writing the physics part of the code (than C++) because it's the more popular language for physics. Furthermore, I have more experience with writing in Python than C++.

With all the information I have, I can now make a full requirement specification.

REQUIREMENT SPECIFICATION

User Requirements

Input requirements

- The user selects the angle or direction the rocket is launched at
- The user selects the power the rocket is launched at
- The user is able to enter their name using the keyboard for high scores and to save the game

- The user is able to choose to save the current game or exit the game without saving when they're finished playing
- The user is able to choose whether to start a new game or continue from a saved game when the program is opened, provided that the last game played hadn't already finished

Processing requirements

- The initial motion of the rocket needs to be calculated based on the rocket's power
- The motion of the rocket needs to be calculated after every frame, based on the previous speed and the gravitational pull surrounding the rocket
- The direction the rocket faces needs to be calculated after every frame, based on the previous direction and the gravitational pull surrounding the rocket
- The rocket's projectile needs to be calculated after every frame
- The screen needs to be updated every frame
- The current level or score that the player is at needs to be recorded for high scores or saving the game
- When the game ends, the new score needs to be compared with previous scores to make an up-to-date high score list
- All levels/scores (including the current one) must be written to a file so they are saved after the program is closed
- The size of the spatial bodies to be drawn on the map needs to be calculated
- The maximum power of the rocket needs to be varied (between levels) to account for the varying difficulties of levels
- The destination body needs to be changed every level
- The level needs to be reset when the rocket crashes into a non-destination object
- The next level map is drawn when the rocket crashes into the destination object
- Sounds needs to be able to be turned on and off when the user changes the sound options

Output requirements

- The direction (angle) and the power of the rocket needs to be displayed before the rocket is launched
- The projectile of the rocket is drawn every frame
- The relative densities of the spatial bodies need to be displayed when the map loads to assist the player
- The high scores screen must be displayed at the end of the game
- The current level or score of the player needs to be displayed during the game
- The current level or score of a saved game needs to be displayed before the game starts
- There must be an option to display the instructions of the game

Design requirements

- Every map will have a home planet, destination and a varying number of other spatial bodies
- Before the rocket is launched, the direction of the rocket and the power bar is displayed for the user to control

- Every object will be an animated image
- The colours in the game will be bright and contrasting, for a fun aesthetic
- Every map will have a different layout
- The user must be able to control the initial direction and power of the rocket easily, with few controls
- There must be a home screen with options to view the instructions of high scores, or to start a new game or continue from a saved one
- There must be in-game sounds as well as background music
- There is no limit in the number of levels that can be played in a single game
- The map is a single screen in size
- The name of the game, 'All Systems Are Go', will always be on the screen

System Requirements

The system requirements will be based on what is needed to run Python 3.3, the version of Python that is most familiar to me. The requirements will also be for the amount of memory needed to store and run the game, and the hardware required for the user to be able to play the game.

Software requirements

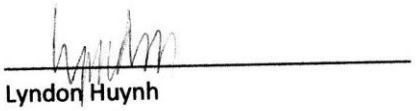
Software	**Justification**
Python 3.3.2	The program required to write the code for the game and the platform the game will run on
Windows XP (or more recent versions of Windows)	The operating system required to run Python 3.3.2

Hardware requirements

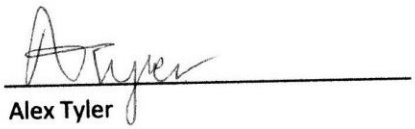
Hardware	**Justification**
Computer with 300MHz (or higher) processor	Clock speed required to run Windows XP
60MB hard disk space	50MB needed to download and store Python 3.3, and up to 10MB to store my game
64MB RAM (or higher)	RAM required to run Windows XP and Python 3.3
Monitor with Super VGA (800 x 600)or higher resolution	To display the game in an adequate resolution
Speakers	To output the game's sounds and music
Computer mouse	To navigate the game
Computer keyboard	To enter the player's name for high scores

Review of Requirements

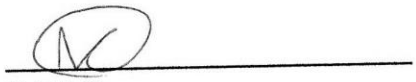
I have read and agreed to the user requirements enclosed.



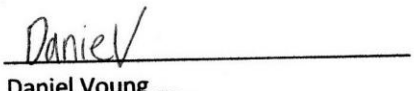
Lyndon Huynh



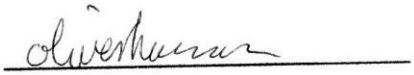
Alex Tyler



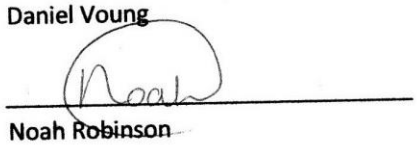
Nakhil Chana



Daniel Voung



Oliver Hassan



Noah Robinson

REFERENCES

http://reviewlagoon.com/review-pocket-tanks-android/

http://stackoverflow.com/questions/3791674/whats-the-best-language-for-physics-modeling

http://www.ics.uci.edu/~pattis/common/handouts/pythoneclipsejava/python.html

SECTION 2

NATURE OF THE SOLUTION

DESIGN SPECIFICATION

The table below shows how my input, processing and output requirements are turned into design objectives.

Requirement	**Objective**
Input requirements	
The user selects the angle or direction the rocket is launched at	There'll be a pointer on the screen to mark the direction from Earth at which the rocket will be launched. The direction is controlled by moving the cursor
The user selects the power the rocket is launched at	There'll be a power bar on the screen to mark the rocket's launch power. The power is controlled by holding down the left mouse key
The user is able to enter their name using the keyboard for high scores and to save the game	When the game has ended, the screen will change to an input screen showing the player's score, where the player enters their name. Then, the high score page or start page will show
The user is able to choose to save the current game or exit the game without saving when they're finished playing	On the screen whilst playing the game, there will be buttons for the user to click to exit the game. If the user chooses to save the game, the name input screen will be displayed
The user is able to choose whether to start a new game or continue from a saved game when the program is opened, provided that the last game played hadn't already finished	On the start page, there will be a button to choose to start playing the game. When that button is clicked, there will be two more buttons for the user to choose whether to start new or continue
Processing requirements	
The initial motion of the rocket needs to be calculated based on the rocket's power	The initial motion will be calculated using the rocket's launch power and the direction of launch
The motion of the rocket needs to be calculated after every frame, based on the previous power and the gravitational pull surrounding the rocket	There'll be an 'update' procedure, containing the calculations of the rocket's power, which will be called upon every frame
The direction the rocket faces needs to be calculated after every frame, based on the previous direction and the gravitational pull surrounding the rocket	There'll be an 'update' procedure, containing the calculations of the rocket's direction, which will be called upon every frame. The gravitational effect that the rocket feels will be updated too
The rocket's projectile needs to be calculated after every frame	The projectile will be calculated using the rocket's power and direction. The calculations will be in an 'update' procedure

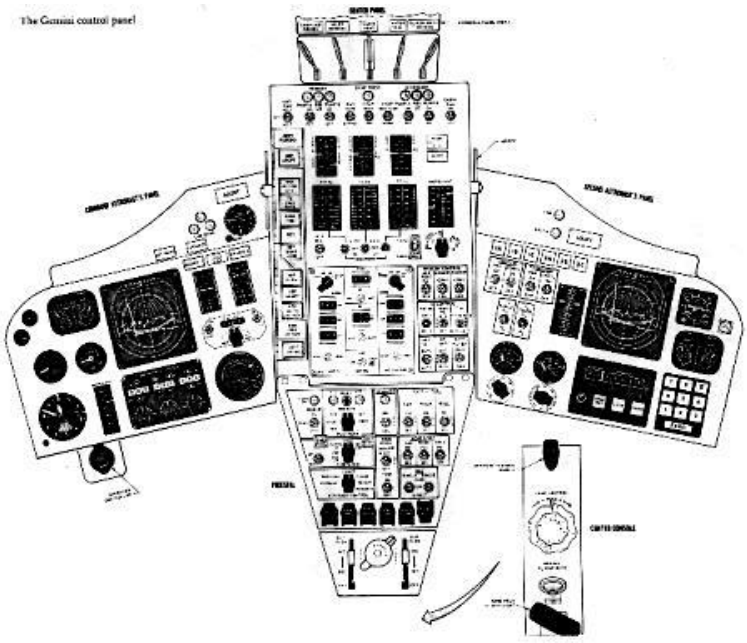
The screen needs to be updated every frame	There'll be a 'draw' procedure, containing all the objects which need to be rendered on to the screen, which will be called upon every frame
The current level or score that the player is at needs to be recorded for high scores or saving the game	The current level that the player is at will be displayed at the side of every map. It will be incremented after every level
When the game ends, the new score needs to be compared with previous scores to make an up-to-date high score list	The score will be the last level the player got to before they lost. The score will be displayed as the user enters their name, and if high enough, their score will be shown on the next page – the high score page
All levels/scores (including the current one) must be written to a file so they are saved after the program is closed	The score will be written alongside a corresponding name, to show who's game the saved game belongs to
The size of the spatial bodies to be drawn on the map needs to be calculated	…based on their calculated gravitational attractions
The maximum power of the rocket needs to be varied (between levels) to account for the varying difficulties of levels	This can be done by changing the values for minimum and maximum power for each level. However, the values will not be displayed, as this will only confuse the player
The destination body needs to be changed every level	To have more variety in each level. Every destination will be a planet, and the planet on the map will be chosen at random
The level needs to be reset when the rocket crashes into a non-destination object	When this happens, the player loses a 'life'. The number of lives will be displayed throughout play
The next level map is drawn when the rocket crashes into the destination object	The next map 'plan' will be retrieved and drawn, and the level will be updated
Sounds needs to be able to be turned on and off when the user changes the sound option	There will be buttons on the start menu as well as in the game to turn the sound on or off
Output requirements	
The direction (angle) and the power of the rocket needs to be displayed before the rocket is launched	The direction will be displayed as a dot (which will become a rocket when launched) and the power will be represented by a power bar
The projectile of the rocket is drawn every frame	…Which will appear to the user as a smooth flight of the rocket
The relative densities of the spatial bodies need to be displayed when the map loads to assist the player	This will be displayed as a number in the middle of the body
The high scores screen must be displayed at the end of the game	The high scores will show the player's name with the corresponding score (level). There will be five high score places

The current level or score of the player needs to be displayed during the game	The current level that the player is at will be displayed at the side of every map. It will be incremented after every level
The current level or score of a saved game needs to be displayed before the game starts	This will be displayed alongside a name, to help identify which game it is
There must be an option to display the instructions of the game	The option will be on the start screen. There will be a button to view the instructions before starting the game.
Design requirements	
Every map will have a home planet, destination and a varying number of other spatial bodies	
Before the rocket is launched, the direction of the rocket and the power bar is displayed for the user to control	
Every object will be an animated image	
The colours in the game will be bright and contrasting, for a fun aesthetic	
Every map will have a different layout	
The user must be able to control the initial direction and power of the rocket easily, with few controls	
There must be a home screen with options to view the instructions of high scores, or to start a new game or continue from a saved one	
There must be in-game sounds as well as background music	
There is no limit in the number of levels that can be played in a single game	
The map is a single screen in size	
The name of the game, ‘All Systems Are Go’, will always be on the screen	

INTERFACE DESIGNS

The Theme

Because the game is called, “All Systems Are Go”, I will design the theme of the game around spacecraft control. I found some images of spacecraft control panels for ideas.



Gemini spacecraft panel outline: http://www.astronautix.com/graphics/g/gempan50.jpg

The control panel consists of rectangular blocks of various sizes and a contrasting colour to the backboard. There are also a couple of circular controls and screens.

Flight deck of Nasa737: http://www.nasa.gov/centers/langley/images/content/70420main_L74-07810.JPG

There are a lot more circular dials on a flight deck than control panel. Spacecraft colours can be seen with this image.

Playfield Design

Based on the images in the section above, I will design the interface of the playfield like this:

The asteroids
Each asteroid will be a different size, which will correlate with their densities. If the player crashes into an asteroid, the number of lives remaining will decrement by 1, and the rocket will be reset. Each level will have a different number of asteroids on the map.

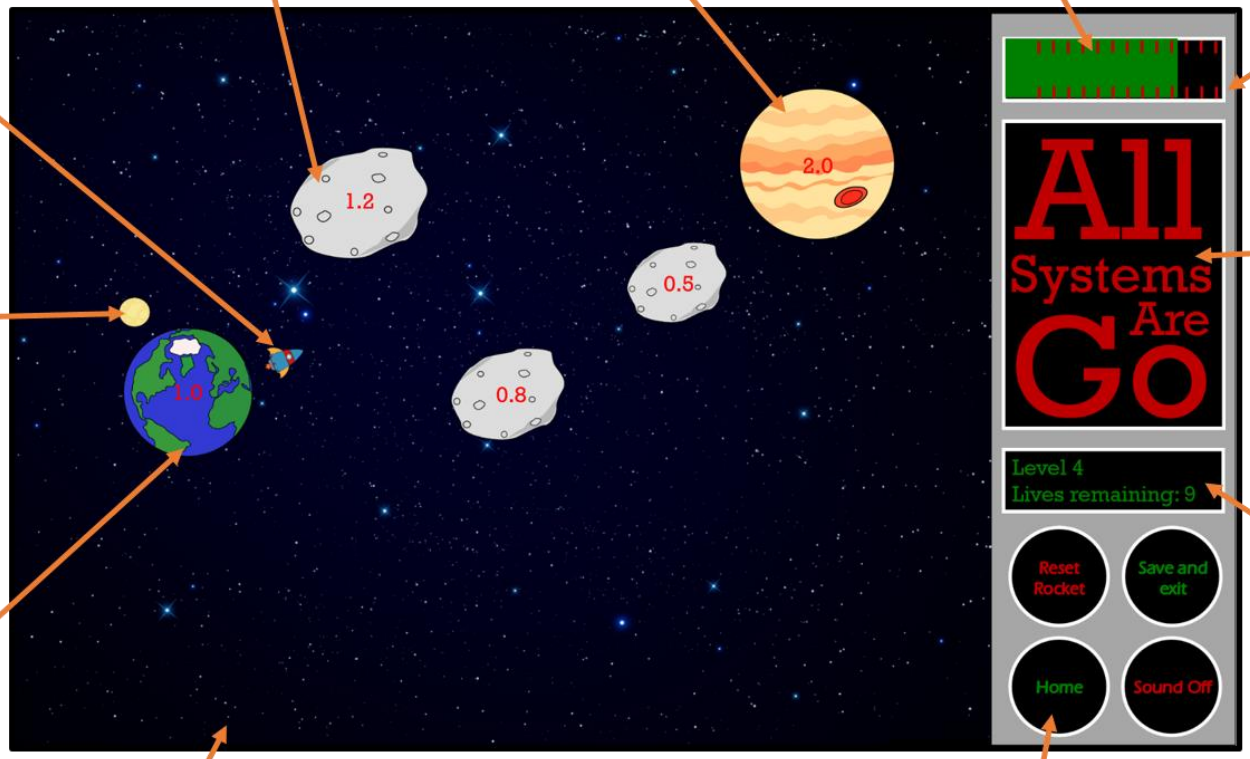
The goal planet
This is the planet the rocket must land on. The actual planet will change between levels – but they will be real planets (Venus, Jupiter, Saturn, Mars…). If the rocket lands on the goal planet, the level increases by 1 and the next map is drawn.

The power bar
As the user holds down the left mouse key, the length of the green bar will increase to maximum, then decrease to minimum over and over again. This gives the player an indication to the rocket's launch power. The red lines are only to add to the control panel aesthetics.

The rocket
The rocket has no gravitational pull. It will point in the direction of travel, which will be updated after every frame. Before launch, the rocket will be represented by a circle, and will only appear at and after launch.

The moon
The moon will be constantly orbiting the Earth unless the rocket crashes into it. The moon will have no gravitational pull, but will give the player an extra life if they crash into it. The moon will come back after every level.

The Earth
This is where the rocket will be launched from. The Earth has a density too, so the rocket may fall back on to it, which will make the player lose a life.



The control panel
If the user clicks anywhere on the control panel, the rocket's power will not be triggered.

The title
The name of the game will be on every screen, but it could not be placed on the playing field, as this would obstruct the player's view. So it is made to be another part of the control panel. Clicking on this will do nothing.

The status bar
Shows the current status of the player (not the rocket). Every time the rocket lands on the planet, the 'level' value is incremented by 1. Every time the rocket crashes into an asteroid, the lives remaining is decremented by 1.

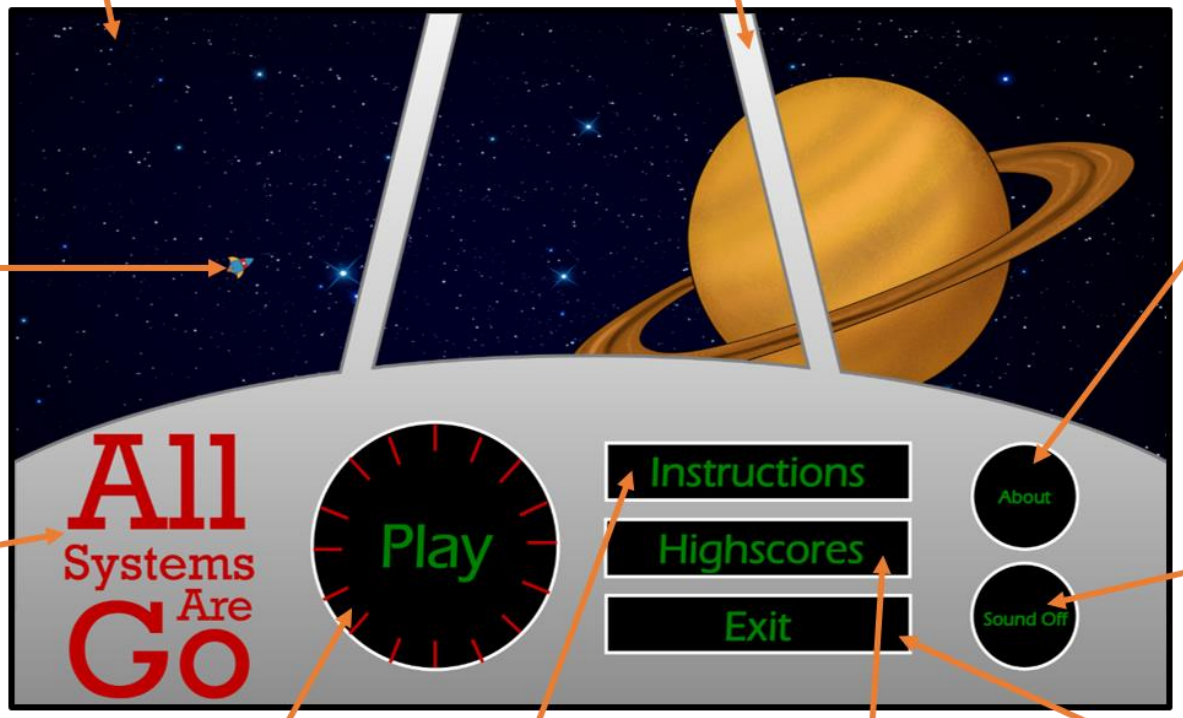
The playing field
If the user clicks anywhere on the playing field, the rocket's power will be triggered. The rocket's direction will follow wherever the cursor is. All the objects on the playing field will be images taken from the internet.

The buttons
- Reset Rocket: If the user realises after the rocket is launched that they won't make it to the planet, instead of waiting they can click this button and lose 1 life.
- Save and exit: The player can save the game so they can continue at the current level with the same number of lives. Clicking this will prompt the name input screen.
- Home: clicking this will take the user back to the home screen. This will not save the game.
- Sound On/Off: If there is music playing, the button will display 'Sound Off', to tell the player to click this if they want the sound off. If there is no sound, the button will display 'Sound On' to indicate "turn the sound on".

The background
This is the image of the stars, which will be drawn onto the screen before anything else.

The control panel
The grey control panel, along with the image of Saturn, will be saved as an image which will be drawn onto the screen. This will be separate to the background image so that the rocket will appear to fly between the panel and the sky.

The rocket
As long as the player stays on the home screen, there will be a rocket which flies across the screen every once in a while. This is to add animation to the screen. The player won't have control of the rocket.



The About button
Clicking this button will show a page with extra information about the page such as where the images came from, what music is playing…

The title
The name of the game will be on every screen. The typeface is Rockwell.

The Sound On/Off button
If there is music playing, the button will display 'Sound Off', to tell the player to click this if they want the sound off. If there is no sound, the button will display 'Sound On' to indicate "turn the sound on".

The Play button
Clicking this button will lead the player to choose whether to start a new game or to load an existing one. The red lines on the button will constantly rotate to make the control panel look more 'alive'.

The Instructions button
Clicking this button will navigate the player to the instructions screen. From there, the user must return to this screen before starting the game.

The Highscores button
Clicking this button will navigate the player to the highscores screen. Each time the button is clicked the file which stores the highscores will be refreshed and displayed.

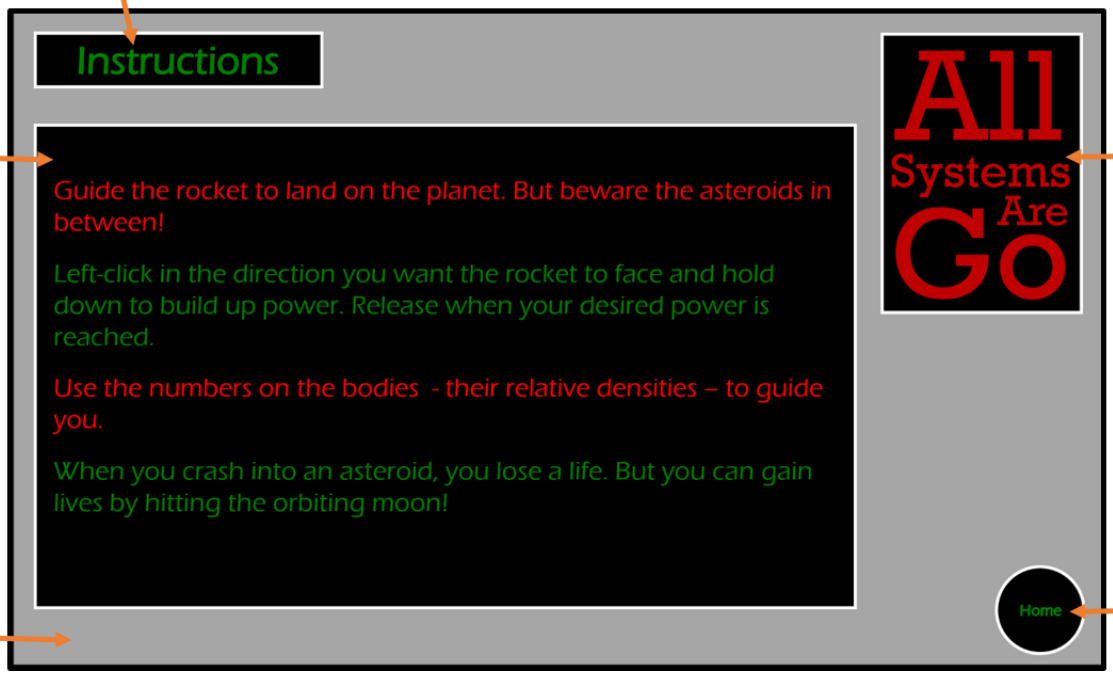
The Exit button
Clicking this button will close the window.

Instructions Screen Design

The heading
The heading is designed in exactly the dame way as the Instructions button on the home screen. This is to make it obvious to the player that what they're seeing is part of the same control panel as the home screen one.

The instructions
The language used is very simple to ensure that everyone can understand the rules. The text colour changes after every sentence to make the instructions easier to read.

The background
The background, a plain grey, will be the default background for all of my screens so that it's not necessary to keep re-drawing images onto the screen.



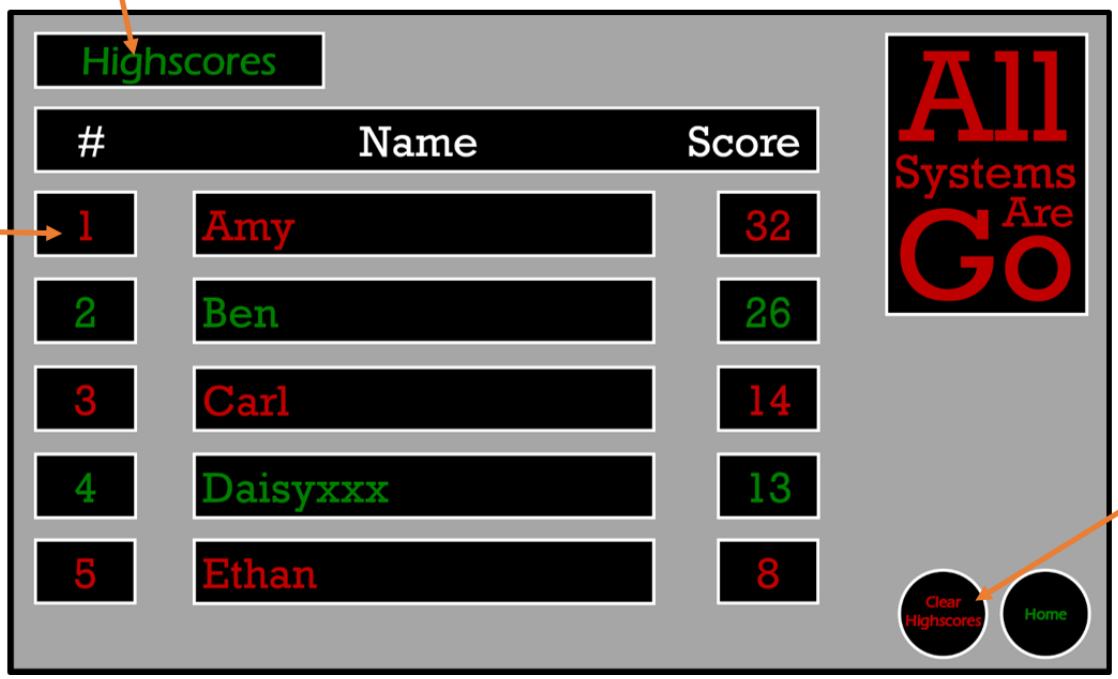
The title
The name of the game will be on every screen. The black box makes the title easier to read as the grey background has the same brightness as the text.

The home button
Clicking this will navigate the user back to the home screen, where they can start the game.

High scores Design

The layout
The title, heading, home button and background are exactly the dame as the instructions screen. But instead of the instructions box there is a highscores table.

The highscores
The names and scores to be displayed will be read from a file which will be updated after every game. The sorting of scores (in descending order) will be done in the file, so that when they need to be displayed on the screen, so more sorting needs being done. This will reduce the amount of time the file will be open for.



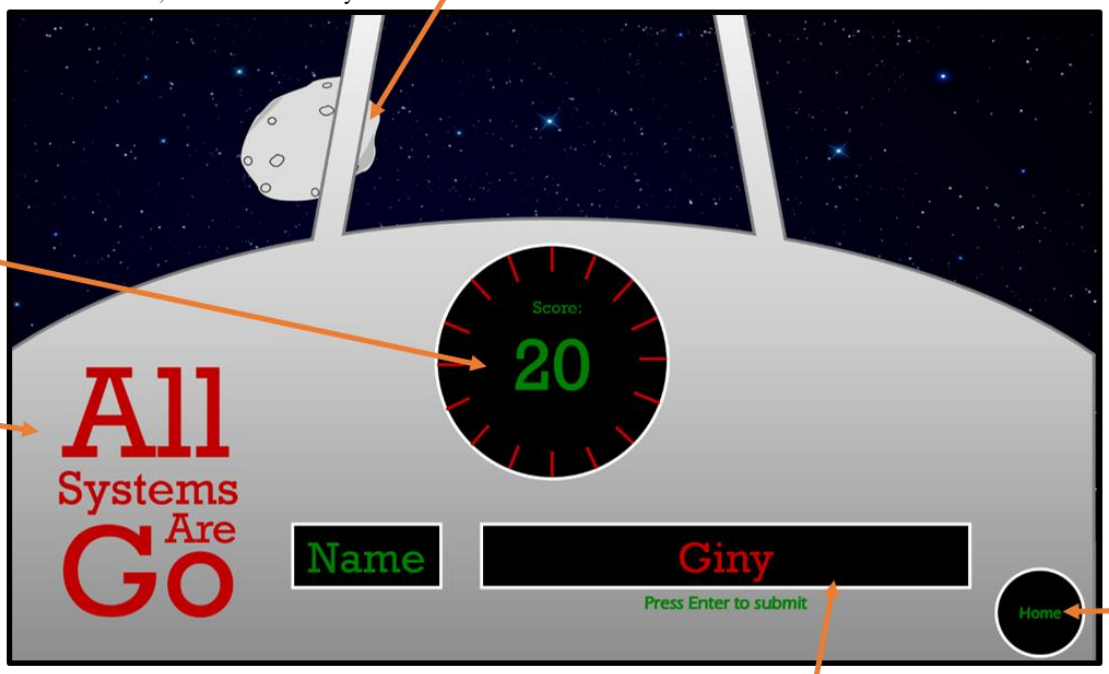
The clear button
Clicking this will delete all the content of the score saving file and the highscores table will be empty.

Name Input Design

The asteroid
In the home screen, there was a rocket which would frequently fly across the screen. In this screen, the rocket will be replaced by an asteroid which will also rotate as it moves across the screen. The change is to add variety to the different screens, which all look fairly similar.

The score
The player's score is the last level the player reached before they ran out of lives. It will be drawn onto the centre of the screen.

The layout
The background, title and control panel follow the same layout here and the home screen.



The home button
Clicking this will navigate the user back to the home screen without the highscore being submitted.

The textbox
Initially this textbox will be blank. But as soon as the player begins to enter their name, the corresponding characters will appear. The box doesn't need to be clicked on before the name is input. The user presses the Enter/Return button on their keyboard to submit their score.

All of these interfaces were designed using Microsoft Office PowerPoint 2013.

Design Review

I have read and agreed to the game designs enclosed.

Lyndon Huynh

Alex Tyler

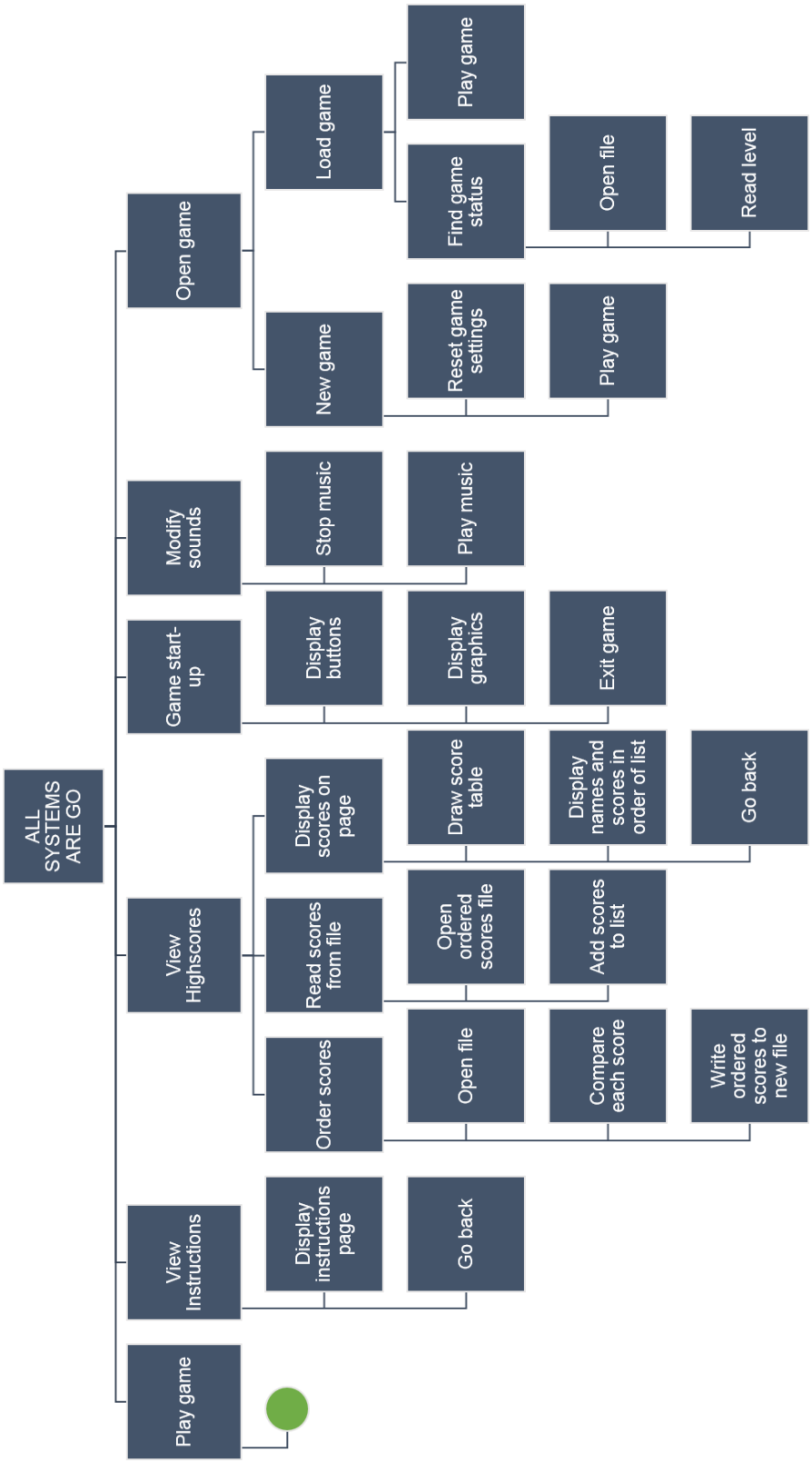
Nakhil Chana

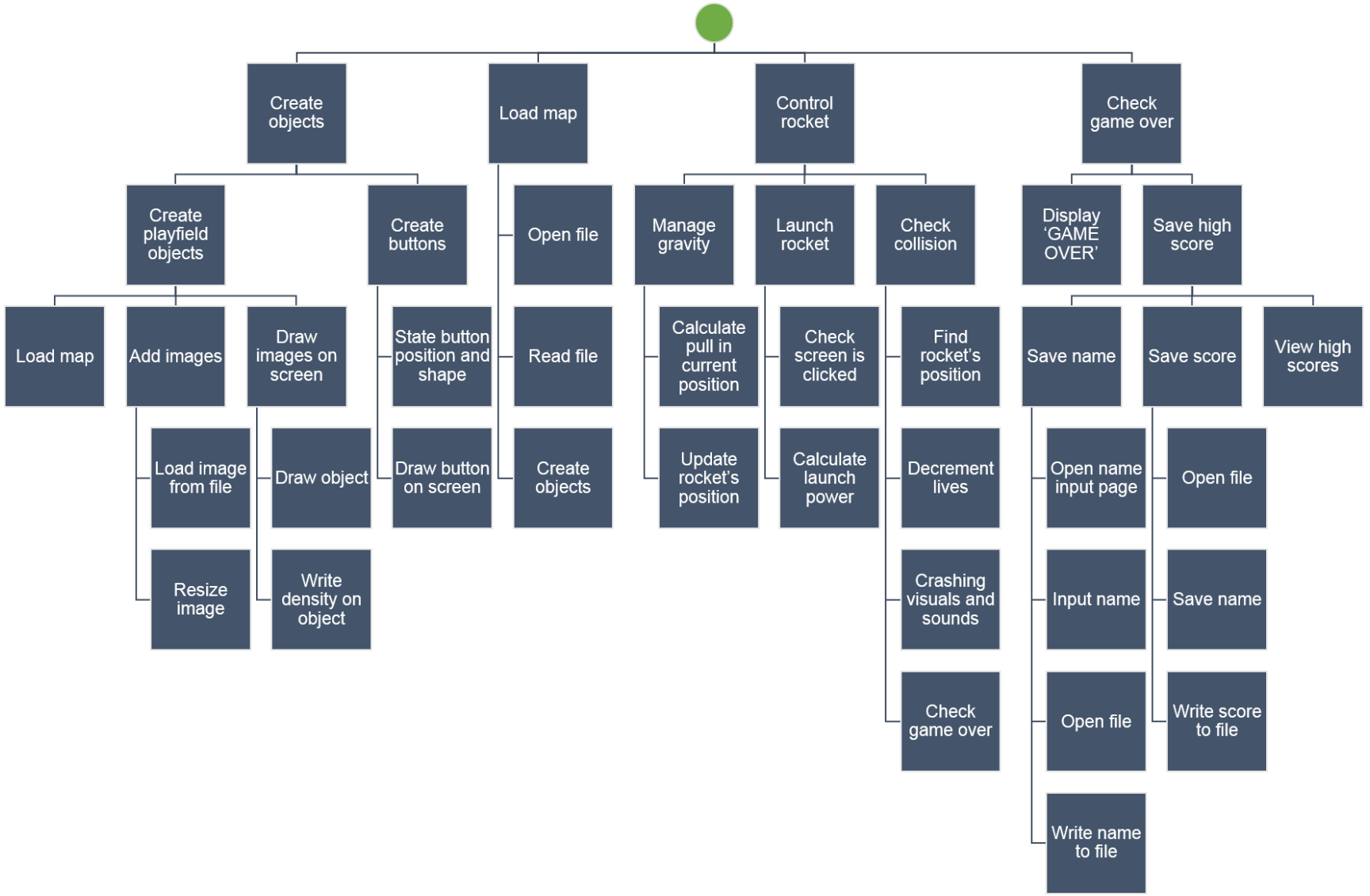
Daniel Voung

Oliver Hassan

Noah Robinson

MODULAR DESIGN – PROCESSES




Create objects
Load map
Control rocket
Check game over
Create playfield objects
Create buttons
Open file
Manage gravity
Launch rocket
Check collision
Display 'GAME OVER'
Save high score
Load map
Add images
Draw images on screen
State button position and shape
Read file
Calculate pull in current position
Check screen is clicked
Find rocket's position
Save name
Save score
View high scores
Load image from file
Draw object
Draw button on screen
Create objects
Update rocket's position
Calculate launch power
Decrement lives
Open name input page
Open file
Resize image
Write density on object
Crashing visuals and sounds
Input name
Save name
Check game over
Open file
Write score to file
Write name to file

DATA STRUCTURES

Since I will be saving games and loading maps, I will need to make files to store things when the program isn't running. However, I had to normalise the files first.

Normalisation

Un-normalised:

- **GAMESFILE(Name, Level, Lives, Highscore)**
  - GAMESFILE stores all games

First Normal Form:

- **GAMESFILE(Name, Level, Lives)**
  - Highscore is the last level the player reaches, so the repeating attribute is removed

Third Normal Form:

- **HIGHSCORESFILE(Name, Level)**
  - HIGHSCORESFILE stores completed games only
- **SAVEDGAMESFILE(Name, Level, Lives)**
  - SAVEDGAMESFILE stores incomplete games only

Files

File name	Description	Sample
SavedGamesFile	To store each game that is saved, with the player's name, current level and number of lives remaining **<NAME>,<LEVEL>,<LIVES>**	('Giny', 5, 5) ('Lyndon', 8, 2)
HighscoresFile	To store the name and high score of all complete games **<NAME>,<LEVEL>**	('Giny', 6) ('Lyndon', 8)
Map(#)	To store the layout of each level. '#' represents an integer from 1 upwards. The number of map files created will be the number of available levels.	Self.asteroids = [[200,100,50,50,1],[400,100,50,50,1]] Self.earth = [100,300,40,20,0.5] Self.goal = [700,300,40,4,0.1]

Variables

Variables to be used in the program:

Variable name	Type	Size (up to)	Description	Sample
Launch_start	Boolean	2	Either a 1 or 0. To mark whether the rocket is in flight or not	0
Speed	Real	9999	The speed of the rocket on the home screen	0.08
Angle	Real	9999	The angle of the rocket on the home screen	Pi
Min_power	Real	9999	The minimum power the rocket can have for a certain level	60

Max_power	Real	9999	The maximum power the rocket can have for a certain level	10
Power	Real	9999	The rocket's current power	10
Page	String	20	The current page being displayed	"home"
Name	String	10	Holds the player's name for high scores or when saving a game	"Giny"
Level	Integer	number of levels	The player's current level	5
Lives	Integer	10	The players remaining number of lives	5
Endgame	Boolean	2	Whether the game is going on or not	False
F_angle	Real	9999	The angle which the rocket is released	0.1
X	Integer	1000	The x-coordinate of the rocket	500
Y	Integer	600	The y-coordinate of the rocket	300
Lc	Boolean	2	Either a 1 or 0. To mark whether the user has left-clicked or not	1

Arrays

Arrays to be used in the program:

Array name	**Size**	**Description**	**Sample**
Self.asteroids	2D array, size depends on number of asteroids in the level	Holds the coordinates, width, height and density of each asteroid	[[200,100,50,50,1], [400,100,50,50,1]]
Self.earth	5	Holds the coordinates, radius half-radius and density of the earth in each level	[100,300,40,20,0.5]
Self.goal	5	Holds the coordinates, radius half-radius and density of the earth in each level	[700,300,40,20,0.1]
Self.rocket	4	Holds the coordinates and the angle which the rocket is fired, and where the user has clicked on the screen to launch the rocket	[136.9, 315.4, 33.2, 13.8]
Scores	Depends on the number of scores in the high scores table – up to 5	Holds all the names and scores from HighscoresFile so that they can be displayed in the high score page	[('Giny',8),('Lyndon',6)]
Maps	2D array, depends on the number of objects in each map	Holds all the level layout details from the file, Map(#), so that they can be displayed on the screen	[Self.asteroids = [[200,100,50,50,1], [400,100,50,50,1]] Self.earth = [100,300,40,20,0.5]

			Self.goal = [700,300,40,4,0.1]]

Objects

Objects to be used in the program:

Object	Description
Buttons	To make it easy for the user to navigate the program. Every button will be made using a class. The one class will make circular and rectangular buttons
Images	To load and manipulate images saved on the computer
Sounds	To load and insert sounds for rocket explosions and background music

LIMITATIONS

My game has been designed as attentively as possible, but in doing so I have realised some of the limitations that I had to get around.

Hardware limitations:

- In my requirements specification, I have stated that the game needs a monitor with only Super VGA resolution, and I have no requirement for a graphics card. Therefore, the graphics on the game could not be too complex, as this would make the game lag, and therefore be unplayable. This is why the graphics in my game are limited to 2D images manipulated through Python only.

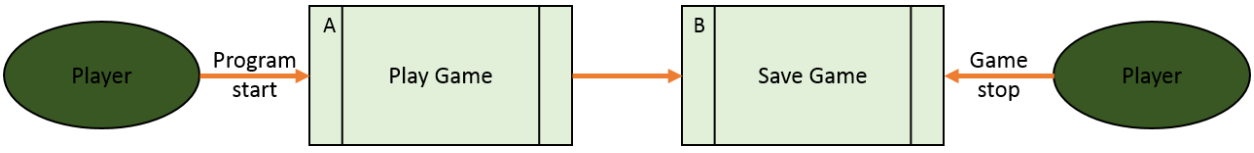
Software limitations:

- Because I will be making the game using Python with the Pygame library:
  - The user will not be able to change the volume of the game, but only turn it on or off
  - The user will not be able to change the size of the window
- Because I will be using simple text files to store data outside the game:
  - There cannot be infinite levels to play – the combined file sizes would take up too much disk space
  - The data will not be encrypted – the files will be continuously manipulated as the game is running

ALGORITHMS

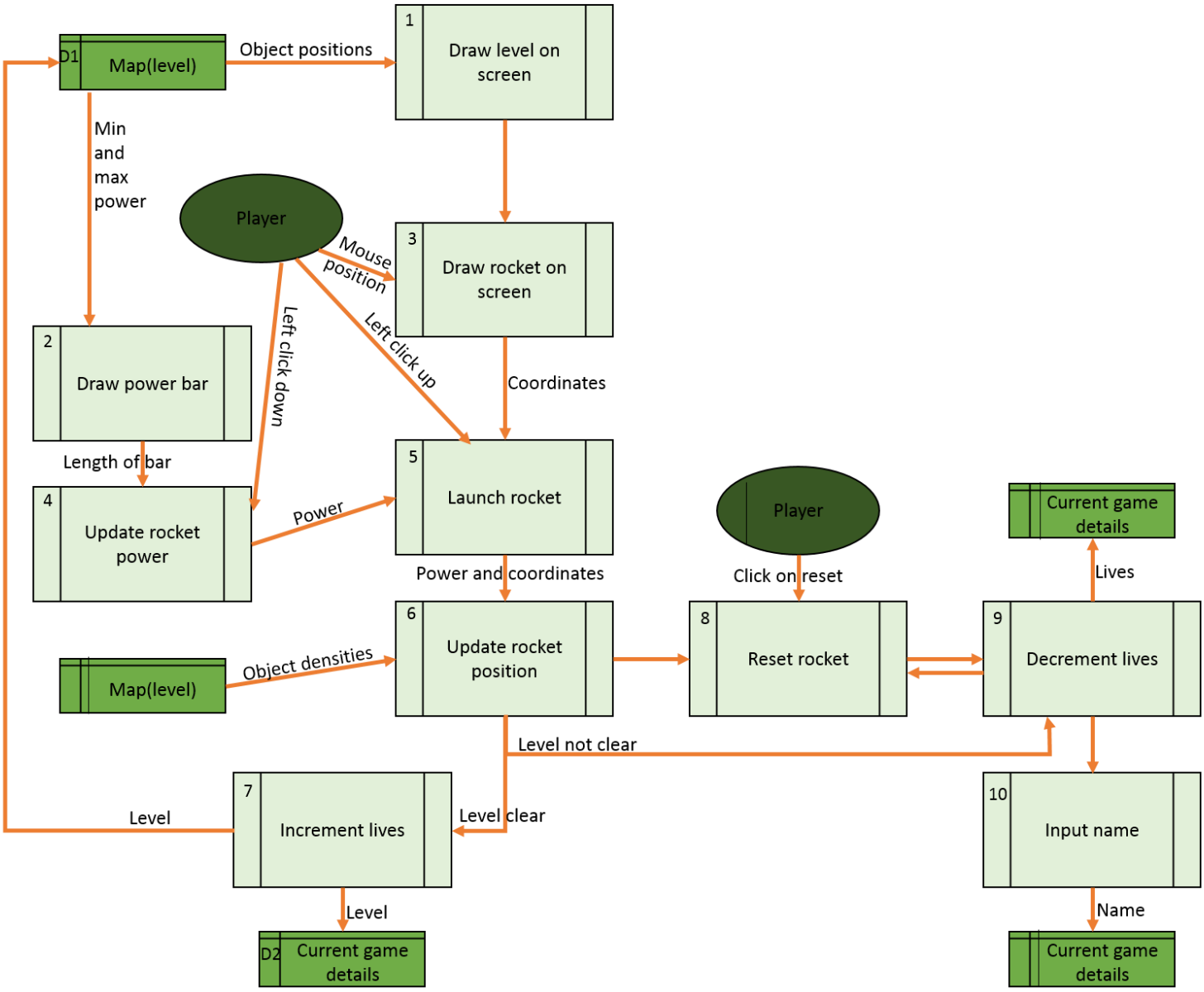
DATA FLOW DIAGRAMS

Level 0 DFD



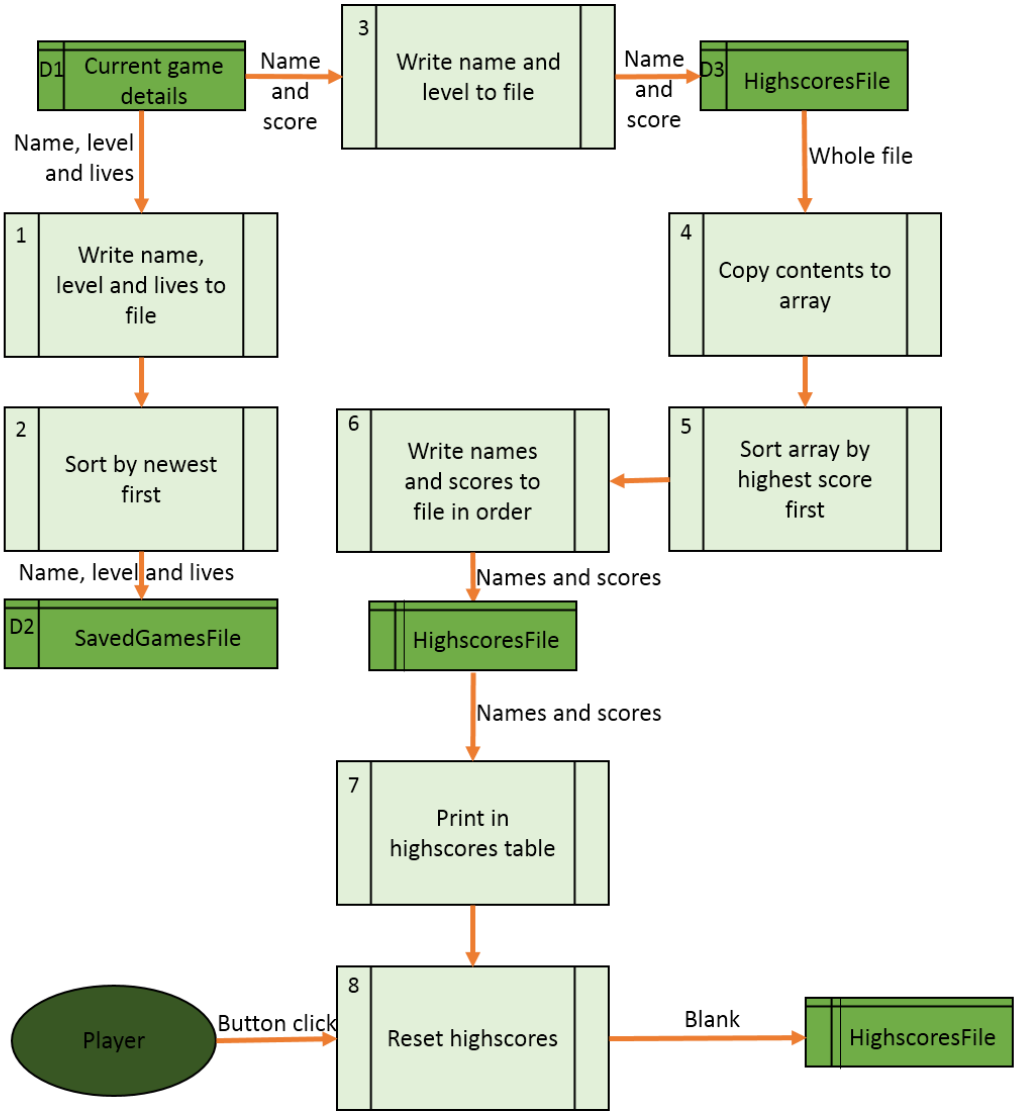
Level 1 DFDs

A: Gameplay:



B: Saving:

The player can save the game at any point or when the game has ended, their game will be added to the high scores. All the data added in to D2: Current game details, will be used when storing games, as shown in the following DFD:



PSEUDOCODE

Below is the pseudocode for All Systems Are Go. Wherever there is a number beside a line (e.g. ……1), it represents a jump to another subroutine, which is labelled by its number and colour.

Main

- Start
- Page = homepage
    - While not quit game:
        - Check for user input 1
        - If on homepage:
            - Draw background and buttons
            - Draw rocket
            - Calculate new rocket position
        - If on load page:

- Draw background
- Draw button for new game
- Draw button for load game

- If on select game page:
  - Draw background
  - Draw table of games
  - Read SavedGamesFile
  - Output names and levels into table
- If on playfield page:
  - Read map(level)
  - Draw background and buttons
  - Draw playing field 2
  - If mouse clicked:
    - Find position clicked
    - Manage rocket launch 3
  - Player crash 4
  - If crashed:
    - Decrement lives
    - Reset level
  - If level complete:
    - Increment level
  - If lives < 0:
    - Freeze display
    - While user not clicked screen:
      - Output “Game Over” on center of screen
    - Page = name input page
- If on name input page:
  - Draw background and button
  - Draw asteroid
  - Calculate new asteroid position
  - Output name
- If on instructions page:
  - Draw background
  - Output instructions
- If on about page:
  - Draw background
  - Output game details
- If on high score page:
  - Draw background and buttons
  - Read HighscoresFile
  - Create score table 5
- Flip display
- Wait 20 milliseconds

- Close game

- End

Check for user input 1

- for any event:
  - if event type = quit game:
    - quit game
- if event type = key down:
  - if key = escape
    - quit game
  - if page = name input:
    - if key = backspace:
      - if length of name > 0:
        - remove last letter from name
    - else if key = return:
      - if game ended:
        - page = high scores
        - Add high scores 1
      - else:
        - page = home
        - write name, level and lives to SavedGamesFile
    - else:
      - if key = a letter key:
        - if length of name < 10:
          - name = name + letter
- if event type = mouse button up:
  - get mouse position
  - if clicked on home button:
    - page = home
  - else:
    - if page = home:
      - if clicked on play button:
        - page = load game page
      - if clicked on instructions button:
        - page = instructions page
      - if clicked on highscores button:
        - page = high scores page
      - if clicked on exit button:
        - quit game
      - if clicked on sound button:
        - if sound currently on:
          - turn sound off
        - else:
          - turn sound on
      - if clicked on about button
        - page = about page
    - else if page = load game page:
      - if clicked on new game button:
        - page = playfield page

- if clicked on load game button:
    - page = select game page
- else if page = select game page:
    - for each saved game:
        - if clicked on that game:
            - read name, level and lives
            - load map(level)
            - page = playfield page
- else if page = playfield page:
    - if clicked on reset button:
        - reset rocket
        - decrement lives
    - if clicked on save button:
        - page = name input page
    - if clicked on sound button:
        - if sound currently on:
            - turn sound off
        - else:
            - turn sound on
- else if page = high scores page:
    - if clicked on clear button
        - clear HighscoresFile

Add high scores 1

- write name and score to HighscoresFile
- copy contents of HighscoresFile to a list
- sort list by score
- while list has more than 5 scores:
    - delete lowest score
- write ordered names and scores to HighscoresFile

Draw playing field 2

- read map(level)
- draw earth using properties from map(level)
- draw asteroids using properties from map(level)
- draw goal planet using properties from map(level)
- draw moon with centre of orbit at earth's coordinates
- calculate new moon position
- if player has fired:
    - draw rocket
    - update rocket position 1

Update rocket position 1

- calculate gravity for earth, asteroids and planet
- apply gravity for earth, asteroids and planet
- if rocket collides with earth, asteroid or screen edge:
    - return crash

- if rocket collides with goal planet:
    - return level complete

Manage rocket launch 3

- If player clicked within playfield:
    - power += power direction
    - calculate rocket power

Player crash 4

- if crashed:
    - play crash sound
    - draw explosion
    - flip display

Create score table 5

- draw score table
- for the number of high scores:
    - display name and scores from HighscoresFile

LOCAL AND GLOBAL VARIABLES

All of the variables to be used in the main section of the code will be a global variable. This will avoid having to cite it as a parameter in all of the other subroutines. Local variables will be used in the subroutines so that they will be able to be reused, and so that values from one subroutine don't accidentally get passed into another. Tis will also make the code cleaner.

TESTING ALGORITHMS

To test the pseudocode in the section above, I shall do white box testing by doing a dry run. The situation shall be:

1. When the game opens on the homepage the user will click the 'Play' button

Line #	Line	Notes
1	Start	
2	Page = homepage	Page = homepage
3	While not quit game:	Quit = FALSE
4	Check for user input	Jump to 'Check for user input' subroutine
55	For any event:	There has been no event, so return to end of line 4
5	If on homepage:	TRUE
6	Draw background and buttons	All the buttons, including 'Play' are displayed
7	Draw rocket	This is the rocket that will periodically fly across the screen
8	Calculate new rocket position	
9, 13, 18, 36, 41, 44, 47	If on load/select game/playfield/name input/instructions/about/high score page	FALSE
51	Flip display	Screen is updated
52	Wait 20 milliseconds	…before going back to the start of the main loop
3	While not quit game:	Quit = FALSE
4	Check for user input	Jump to 'Check for user input' subroutine
55	For any event:	The user has clicked on the 'Play' button
56	If event type = quit game:	FALSE
58	If event type = key down:	FALSE
76	If event type = mouse button up:	TRUE
77	Get mouse position	This will be the coordinates of the 'Play' button
78	If clicked on home button:	FALSE
80, 81	Else: If page = home:	TRUE
82	If clicked on Play button:	TRUE
83	Page = load game page	Page = load game page
84, 86, 88, 90, 95	If clicked on instructions/highscores/exit/sound/about button:	FALSE. End of subroutine
3	While not quit game:	Quit = FALSE
4	Check for user input	Jump to 'Check for user input' subroutine
55	For any event:	There has been no event, so return to end of line 4
5	If on homepage:	FALSE
9	If on load page:	TRUE
10	Draw background	Load game page is displayed

11	Draw button for New Game	
12	Draw button for Load Game	
13, 18, 36, 41, 44, 47	If on select game/playfield/name input/instructions/about/high score page	FALSE
51	Flip display	Screen is updated
52	Wait 20 milliseconds	…before going back to the start of the main loop

2. The user will load a game belonging to 'Bob', which is currently on level 8 with 0 lives remaining

Line #	Line	Notes
3	While not quit game:	Quit = FALSE
4	Check for user input	Jump to 'Check for user input' subroutine
55	For any event:	The user has clicked on the 'Load Game' button
56	If event type = quit game:	FALSE
58	If event type = key down:	FALSE
76	If event type = mouse button up:	TRUE
77	Get mouse position	This will be the coordinates of the 'Load Game' button
78	If clicked on home button:	FALSE
80, 81	Else: If page = home:	FALSE
97	Else if page = load game page:	TRUE
98	If clicked on new game button:	FALSE
100	If clicked on load game button:	TRUE
101	Page = select game page	Page = select game page. End of subroutine
3	While not quit game:	Quit = FALSE
4	Check for user input	Jump to 'Check for user input' subroutine
55	For any event:	There has been no event, so return to end of line 4
5	If on homepage:	FALSE
9	If on load page:	FALSE
13	If on select game page:	TRUE
14	Draw background	Saved games page is displayed
15	Draw table of games	
16	Read SavedGamesFile	The output names and levels are from SavedGamesFile
17	Output names and levels into table	
18, 36, 41, 44, 47	If on playfield/name input/instructions/about/high score page	FALSE
51	Flip display	Screen is updated
52	Wait 20 milliseconds	…before going back to the start of the main loop
3	While not quit game:	Quit = FALSE
4	Check for user input	Jump to 'Check for user input' subroutine
55	For any event:	The user has clicked on Bob's game

56	If event type = quit game:	FALSE
58	If event type = key down:	FALSE
76	If event type = mouse button up:	TRUE
77	Get mouse position	This will be the coordinates of the row in the table with Bob's game
78	If clicked on home button:	FALSE
80, 81	Else: If page = home:	FALSE
97	Else if page = load game page:	FALSE
102	Else if page = select game page:	TRUE
103	For each saved game:	Go through all saved games
104	If clicked on that game:	Only clicked on Bob's game
105	Read name, level and lives	Name = Bob, level = 8, lives = 0
106	Load map(level)	The game will start with map(8)
107	Page = playfield page	Page = playfield page. End of subroutine
3	While not quit game:	Quit = FALSE
4	Check for user input	Jump to 'Check for user input' subroutine
55	For any event:	There has been no event, so return to end of line 4
5, 9, 13	If on home/load/select game page:	FALSE
18	If on playfield page:	TRUE
19	Read map(8)	The file will contain positions and densities of objects
20	Draw background and buttons	Playfield page (without objects in playfield) is displayed
21	Draw playing field	Jump to 'Draw playing field' subroutine
128	Read map(8)	The file will contain positions and densities of objects
129	Draw earth	…using properties from map(8)
130	Draw asteroids	
131	Draw goal planet	
132	Draw moon with center of orbit at earth's coordinates	Coordinates given in map(8). Now complete playfield page is displayed
133	Calculate new moon position	The moon will continuously orbit the earth
134	If player has fired:	FALSE. End of subroutine, so return to end of line 21

3. The user will play the game. They will:
   - land on the goal planet and go to the next level
   - crash into an asteroid and lose a life
   - click on the window when 'Game Over' is displayed

Line #	Line	Notes
22	If mouse clicked:	TRUE

23	Find position clicked	This is where the user clicks on the playfield to launch the rocket
24	Manage rocket launch	Jump to 'Manage rocket launch' subroutine
143	If player clicked within playfield:	TRUE
144	Power = power + direction	Power given by the length of the power bar
145	Calculate rocket power	End of subroutine, so return to end of line 24
25	Player crash	Jump to 'Player crash' subroutine
146	If crashed:	FALSE (the rocket hasn't been fired yet). End of subroutine
26, 29, 31	If crashed/level complete/ lives <0:	FALSE
36, 41, 44, 47	If on name input/instructions/about/high score page	FALSE
51	Flip display	Screen is updated
52	Wait 20 milliseconds	…before going back to the start of the main loop
3	While not quit game:	Quit = FALSE
4	Check for user input	Jump to 'Check for user input' subroutine
55	For any event:	There has been no event, so return to end of line 4
5, 9, 13	If on home/load/select game page:	FALSE
18	If on playfield page:	TRUE
19	Read map(8)	The file will contain positions and densities of objects
20	Draw background and buttons	Playfield page (without objects in playfield) is displayed
21	Draw playing field	Jump to 'Draw playing field' subroutine
128	Read map(8)	The file will contain positions and densities of objects
129	Draw earth	…using properties from map(8)
130	Draw asteroids	
131	Draw goal planet	
132	Draw moon with center of orbit at earth's coordinates	Coordinates given in map(8). Now complete playfield page is displayed
133	Calculate new moon position	The moon will continuously orbit the earth
134	If player has fired:	TRUE
135	Draw rocket	…Instead of red dot
136	Update rocket position	Jump to 'Update rocket position' subroutine
137	Calculate gravity for earth, asteroids and planet	This will make the rocket respond to the objects by travelling in a curved path
138	Apply gravity for earth, asteroids and planet	
139	If rocket collides with earth, asteroid or screen edge:	FALSE

141	If rocket collides with goal planet:	FALSE (the rocket is still travelling). **This whole section of the dry run will be repeated as long as the rocket is still in flight, so I will jump to when the rocket is in contact with an object.**
36, 41, 44, 47	If on name input/instructions/about/high score page	FALSE
51	Flip display	Screen is updated
52	Wait 20 milliseconds	…before going back to the start of the main loop
	…	
141	If rocket collides with goal planet:	TRUE
142	Return level complete	End of subroutine, so return to end of line 25
26	If crashed:	FALSE
29	If level complete:	TRUE
30	Increment level	Level = 9
32	If lives < 0	FALSE
36, 41, 44, 47	If on name input/instructions/about/high score page	FALSE
51	Flip display	Screen is updated
52	Wait 20 milliseconds	…before going back to the start of the main loop
Now that the user is on level 9, map(9) and its properties will be loaded in the same way as map(8). The launching and flight of the rocket will also be the same. Therefore I will skip to the part in the algorithm where the user crashes. From 'Update rocket position' subroutine…		
139	If rocket collides with earth, asteroid or screen edge:	TRUE
140	Return crash	End of subroutine, so return to end of line 25
26	If crashed:	TRUE
27	Decrement lives	Lives = -1
28	Reset level	
29	If level complete:	FALSE
31	If lives < 0:	TRUE
32	Freeze display	Playfield is frozen
33	While user not clicked screen:	Clicked = FALSE
34	Output "Game Over" on center of screen	'Game Over' message displayed. Loop continues until user clicks screen:
33	While user not clicked screen:	Clicked = TRUE
35	Page = name input page	Page = name input page
36	If on name input page:	TRUE
37	Draw background and button	Name input page now displayed
38	Draw asteroid	This is the asteroid that will periodically fly across the screen
39	Calculate new asteroid position	
40	Output name	Name is still 'Bob', so that will be displayed on screen

41, 44, 47	If on instructions/about/high score page	FALSE
51	Flip display	Screen is updated
52	Wait 20 milliseconds	…before going back to the start of the main loop

4. The user will enter their name as 'Bobby' and press return to save the high score

Line #	Line	Notes
3	While not quit game:	Quit = FALSE
4	Check for user input	Jump to 'Check for user input' subroutine
55	For any event:	The user has entered 'b' on the keyboard
56	If event type = quit game:	FALSE
58	If event type = key down:	TRUE
59	If key = escape:	FALSE
61	If page = name input:	TRUE
62	If key = backspace:	FALSE
65	If key = return:	FALSE
72, 73	Else: if key = a letter key:	TRUE
74	If length of name < 10:	TRUE
75	Name = name + letter	Name = 'Bobb'
76	If event type = mouse button up:	FALSE. End of subroutine
3	While not quit game:	Quit = FALSE
4	Check for user input	Jump to 'Check for user input' subroutine
55	For any event:	There has been no event, so return to end of line 4
5, 9, 13,18	If on home/load/select game/playfield page:	FALSE
36	If on name input page:	TRUE
37	Draw background and button	Name input page now displayed
38	Draw asteroid	This is the asteroid that will periodically fly across the screen
39	Calculate new asteroid position	
40	Output name	'Bobb' is displayed
41, 44, 47	If on instructions/about/high score page	FALSE
51	Flip display	Screen is updated
52	Wait 20 milliseconds	…before going back to the start of the main loop
3	While not quit game:	Quit = FALSE
4	Check for user input	Jump to 'Check for user input' subroutine
55	For any event:	The user has entered 'y' on the keyboard
56	If event type = quit game:	FALSE
58	If event type = key down:	TRUE
59	If key = escape:	FALSE
61	If page = name input:	TRUE
62	If key = backspace:	FALSE
65	If key = return:	FALSE

72, 73	Else: if key = a letter key:	TRUE
74	If length of name < 10:	TRUE
75	Name = name + letter	Name = 'Bobby'
76	If event type = mouse button up:	FALSE. End of subroutine
3	While not quit game:	Quit = FALSE
4	Check for user input	Jump to 'Check for user input' subroutine
55	For any event:	There has been no event, so return to end of line 4
5, 9, 13,18	If on home/load/select game/playfield page:	FALSE
36	If on name input page:	TRUE
37	Draw background and button	Name input page now displayed
38	Draw asteroid	This is the asteroid that will periodically fly across the screen
39	Calculate new asteroid position	
40	Output name	'Bobby' is displayed
41, 44, 47	If on instructions/about/high score page	FALSE
51	Flip display	Screen is updated
52	Wait 20 milliseconds	…before going back to the start of the main loop
3	While not quit game:	Quit = FALSE
4	Check for user input	Jump to 'Check for user input' subroutine
55	For any event:	The user has entered the return key on the keyboard
56	If event type = quit game:	FALSE
58	If event type = key down:	TRUE
59	If key = escape:	FALSE
61	If page = name input:	TRUE
62	If key = backspace:	FALSE
65	If key = return:	TRUE
66	If game ended:	TRUE
67	Page = high scores	Page = high scores
68	Add high scores	Jump to 'Add high scores' subroutine
122	Write name and score to HighscoresFile	('Bobby, 9) will be written
123	Copy contents of HighscoresFile to a list	For sorting
124	Sort list by score	With 1 high score, no sorting is needed
125	While list has more than 5 scores:	There are less than 5 scores
127	Write ordered names and scores to HighscoresFile	HighscoresFile is overwritten. End of subroutine, so return to end of line 68
76	If event type = mouse button up:	FALSE. End of subroutine
3	While not quit game:	Quit = FALSE
4	Check for user input	Jump to 'Check for user input' subroutine
55	For any event:	There has been no event, so return to end of line 4

5, 9, 13,18,36, 41, 44	If on home/load/select game/playfield/name input/instructions/about page:	FALSE
47	If on high scores page:	TRUE
48	Draw background and buttons	High scores page displayed
49	Read HighscoresFile	To get all names and scores
50	Create score table	Jump to 'Create score table' subroutine
150	Draw score table	Score table now displayed
151	For number of high scores:	High scores displayed in the score table in score order. End of subroutine, so return to end of line 50
152	Display name and scores from HighscoresFile	
51	Flip display	Screen is updated
52	Wait 20 milliseconds	…before going back to the start of the main loop

5. The user will exit the game by clicking the close button at the top of the window

Line #	Line	Notes
3	While not quit game:	Quit = FALSE
4	Check for user input	Jump to 'Check for user input' subroutine
55	For any event:	The user has clicked exit
56	If event type = quit game:	TRUE
57	Quit game	Window closed

The dry run shows that my algorithms work and meet the design requirements.

TEST STRATEGY

Although white box testing is done, black box testing must be done when the program is complete to show that all the requirements in the specification are met. I must also do user testing to ensure that my end users will be satisfied with the finished product – and if not I will be able to amend the program.

These black box tests will cover normal, extreme and abnormal inputs.

TEST PLAN

Test plan for the finished game:

Requirement tested	Input method	Input	Expected result
Page changes	Mouse click on button with name:	Play	Screen changes to new/load game page
		Instructions	Screen changes to instructions page
		About	Screen changes to about page
		Exit	Game closes

		New game	Screen changes to playfield page
		Load game	Screen changes to saved games page
		Home	Screen changes to homepage
		Save and exit	Screen changes to name input page
		Continue without saving	Screen changes to homepage
	Mouse click on row in saved games table	1st row	Screen changes to playfield page
		2nd row	Screen changes to playfield page
	Press keyboard button:	return	If saving high score: screen changes to high score page If saving unfinished game: screen changes to home page
Sound	Mouse click on button with name:	Sound Off	Sounds and music turns off Button changes to 'Sound On'
		Sound On	Sounds and music turns on Button changes to 'Sound Off'
	Collide rocket with:	An asteroid	Crash sound
		Goal planet	Cheer sound
		Earth	Crash sound
		Screen border	Crash sound
		Moon	Boop sound
Starting a game	Click on button:	New game	Playfield shows Level = 1 Lives remaining = 10
	Click on row in saved games table	Row with : Name: Giny Current level: 3 Lives: 9	Playfield shows Level = 3 Lives remaining = 9 When game finished, name showing: Giny

Rocket launch	Move cursor:	North of Earth	Red dot lies on top of Earth
		East of Earth	Red dot lies on right of Earth
		South of Earth	Red dot lies on bottom of Earth
		West of Earth	Red dot lies on left of Earth
		North-east of Earth	Red dot lies on top right of Earth
	Mouse click and hold down:	On playfield	Power in the power bar goes up and down
		Outside playfield	Nothing happens
	Mouse click up:	On playfield	Power bar freezes Red dot changes to rocket, facing and moving towards cursor
		Outside playfield	Power bar freezes Rocket doesn't appear until cursor moves back into playfield
Rocket Motion	Launch rocket with cursor on:	The goal planet	Rocket initially travels towards goal planet, then curves towards asteroid
		The nearest asteroid	Rocket travels directly to asteroid
		Empty space on playfield	Rocket initially travels towards cursor, then curves towards nearest/most dense object
Rocket collisions (crashing)	Collide rocket with:	The goal planet	Level = level + 1 Map changes Rocket changes to red dot
		An asteroid	Lives = lives – 1 Rocket explodes, then reappears as red dot on Earth
		Earth	Lives = lives – 1

			Rocket explodes, then reappears as red dot on Earth
		Playfield border	Lives = lives – 1 Rocket explodes, then reappears as red dot on Earth
		The moon	Lives = lives + 1 Moon disappears for rest of level Rocket continues same trajectory
Level change	Go to level:	1	Map of playfield displayed corresponds to coordinates and densities on map1 file
		2	Map of playfield displayed corresponds to coordinates and densities on map2 file
		3	Map of playfield displayed corresponds to coordinates and densities on map3 file
		Last level, then land rocket on goal planet	Playfield freezes, 'Game complete' displayed on screen. Name input page displayed once clicked
Game over	When lives = 0:	Crash rocket into asteroid/Earth/border	Playfield freezes. 'Game over' displayed. Name input page displayed once clicked
		Launch rocket, then click 'reset rocket' button	Playfield freezes. 'Game over' displayed. Name input page displayed once clicked
Saving	When level = 2 and lives = 6, click button:	'Save game'	Name input page displayed
	Enter name:	'SaveTest' (then press return key)	Homepage displayed. ('SaveTest', 2, 6) appended to SavedGamesFile

Name input	Press keyboard button:	‘a’	Name bar will display :	‘a’
		‘b’		‘ab’
		‘c’		‘abc’
		‘A’		‘abcA’
		‘B’		‘abcAB’
		‘C’		‘abcABC’
		‘1’		‘abcABC1’
		‘2’		‘abcABC12’
		‘3’		‘abcABC123’
		‘#’		‘abcABC123’
		‘!’		‘abcABC123’
		Backspace		‘abcABC12’
		Space bar, then ‘d’		‘abcABC12 d’
		‘e’		‘abcABC12 d’
		Return	If saving completed game: High scores page displayed HisghscoresFile updated	
			If saving incomplete game: Homepage displayed SavedGamesFile updated	
Highscores	Click on button with name:	Highscores	Name and score displayed in descending order of score. Maximum of five scores displayed	
	Open file	HighscoresFile	Name and level stored in order of highest score first	
Load game	In saved games page, click on entry:	‘SavedTest’, ‘2’	Playfield page displayed Lives = 6, level = 2 Rocket is unlaunched	
		Empty box	Nothing	

REFERENCES

Clipart images on interface designs:

- http://www.clipartlord.com/category/space-clip-art/planets-clip-art/page/3/
- http://www.clker.com/clipart-2123.html
- http://www.mycutegraphics.com/graphics/space-images.html

SECTION 3

DEVELOPMENT

Here I will describe how I made the game, as I make it. I will do white box testing along the way so that only black box testing needs to be carried out on the completed game.

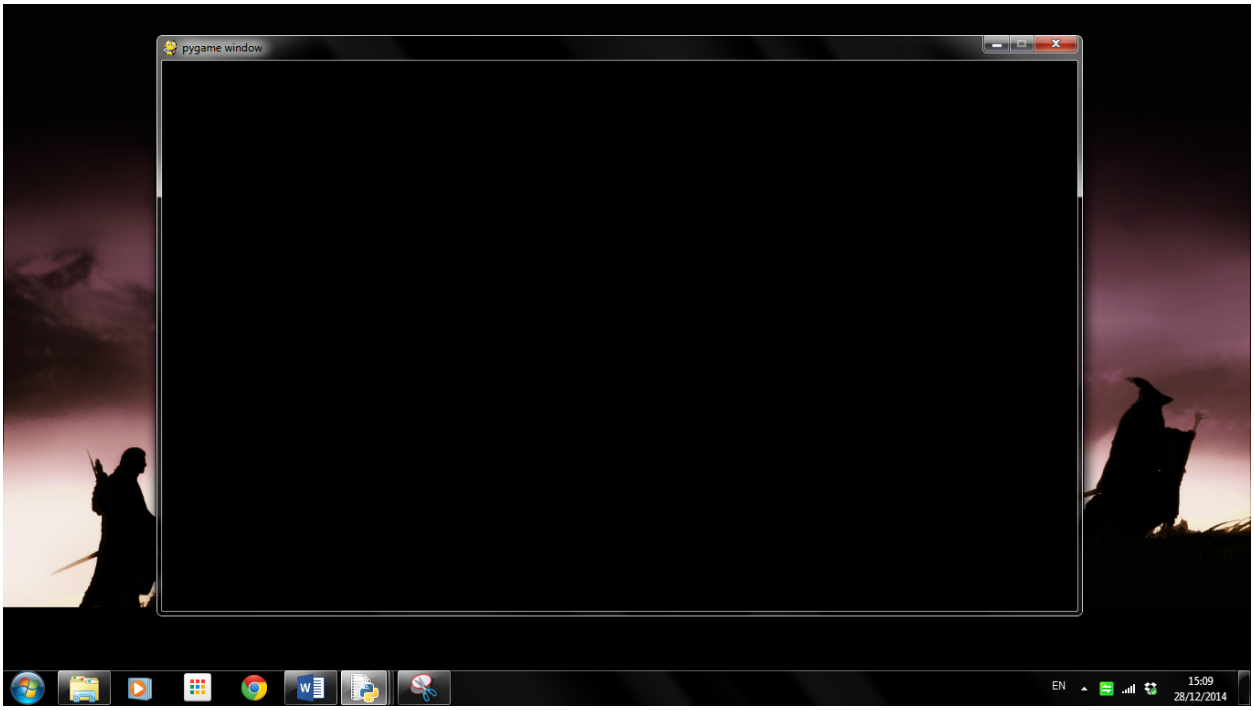
First I imported all the libraries I needed to make the game. Then I declared the size of the window and the size of the playfield and created the window.

```
from pygame import * # for built in Pygame modules
from random import * # to generate random numbers
from math import * # to calculate projectiles
from ast import *#for files
from operator import * #for files

init() # initialise pygame

#create window
game_size = width,height = 800,600
whole_size = maxwidth,height = 1000,600
screen = display.set_mode(whole_size)
```

Then I ran the program to see if the screen size is what I expected.



It is. So then I loaded the sounds. This only needs to be done once throughout the program so I did it here. (Sounds are all royalty free)

```
#sounds
music = mixer.music.load('sounds/Pamgaea.ogg')
crash_sound = mixer.Sound('sounds/general_crash.ogg')
cheer_sound = mixer.Sound('sounds/Eehh.ogg')
death_sound = mixer.Sound('sounds/death_crash.ogg')
moon_sound = mixer.Sound('sounds/moon_crash.ogg')
```

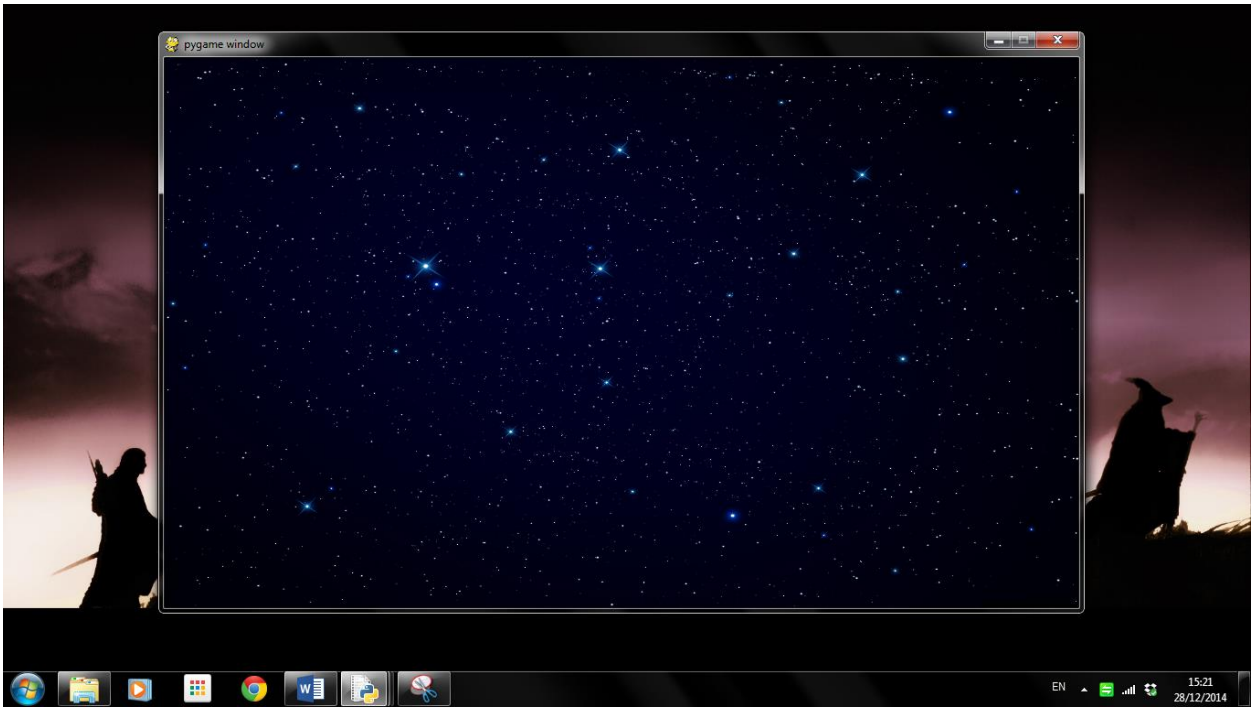
Then I loaded the images. Since some of the images required shrinking, I created a procedure to load and scale the images.

```
#images
def LoadAndScale(path,w,h):
    #loading images from the 'images' folder
    picture = image.load('images/' + path)
    if w != 0 and h != 0:
        #scaling images if I have stated the new width and height
        picture = transform.scale(picture,(w,h))
    return picture
#the images in the folder
background_img = LoadAndScale('background.png',maxwidth,height)
home_img = LoadAndScale('home_background.png',maxwidth,height)
name_img = LoadAndScale('name_input_background.png',maxwidth,height)
moon_img = LoadAndScale('moon.png',20,20)
rocket_img = LoadAndScale('bluerocket.png',20,20)
jupiter_img = LoadAndScale('jupiter.png',0,0)
mars_img = LoadAndScale('mars.png',0,0)
saturn_img = LoadAndScale('saturn.png',0,0)
uranus_img = LoadAndScale('uranus.png',0,0)
venus_img = LoadAndScale('venus.png',0,0)
earth_img = LoadAndScale('earth.png',0,0)
asteroid_img = LoadAndScale('asteroid.png',0,0)
dial_img = LoadAndScale('dial.png',194,194)
explosion_img = LoadAndScale('explosion.png',40,40)
spinning_asteroid = transform.scale(asteroid_img,(100,100))
#generating a random goal planet
random_planet = [jupiter_img, mars_img, saturn_img, uranus_img, venus_img]
the_planet = random_planet[randint(0,4)]
```

To test if I had loaded the sounds and images properly, I added this to the end of the code:

```
mixer.music.play(-1) # test if  music is loaded properly
screen.blit(background_img,(0,0)) #test if images are loaded and scaled properly
display.flip()
```

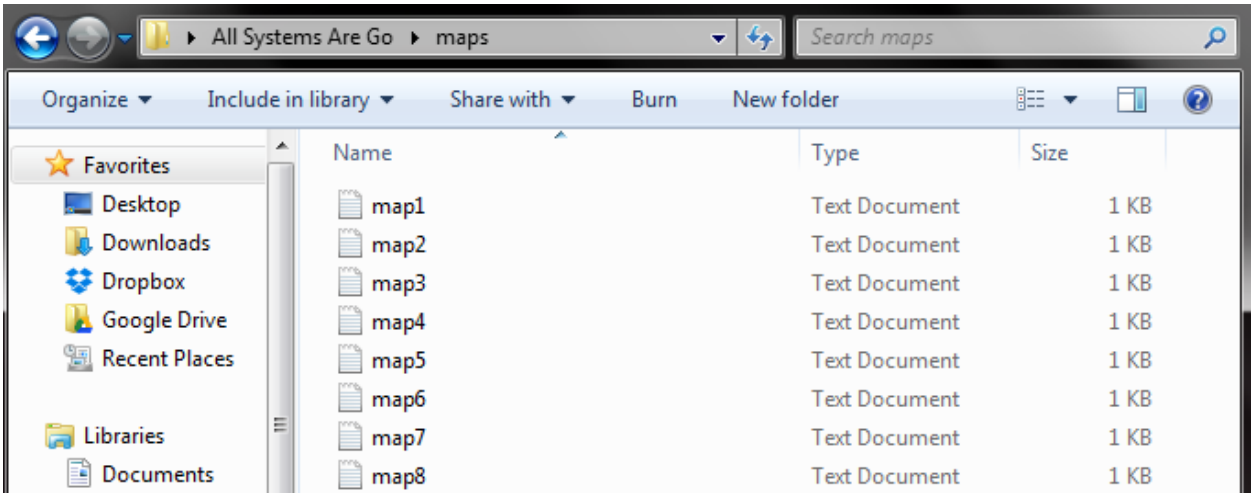
When I ran the program, the music started playing and the window looked like this:



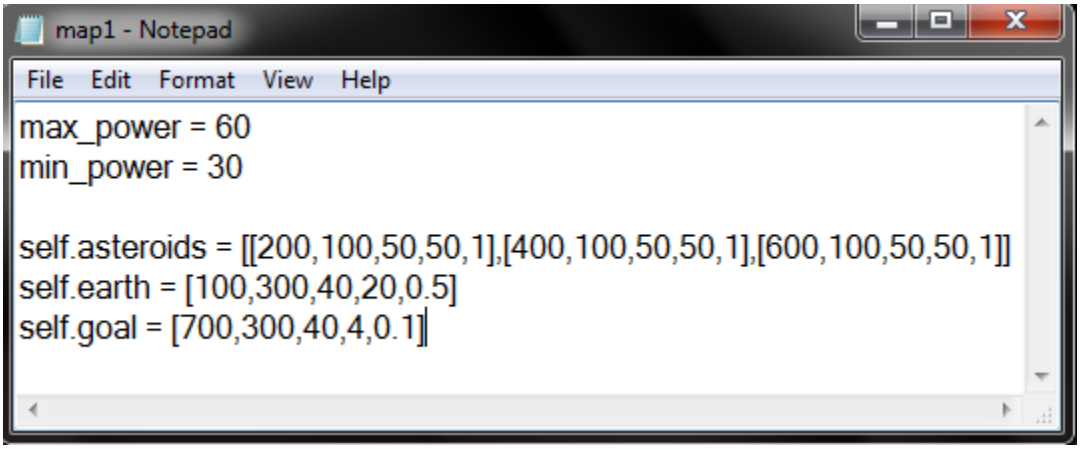
The test was successful. So then I declared the fonts and colours I will be using so that when I do use them, I can reference them by the variable name.

```
#fonts
density_font = font.SysFont("rockwell", 18)
button_font_14 = font.Font('fonts/Futura.ttf',14)
button_font_40 = font.Font('fonts/Futura.ttf',40)
button_font_55 = font.Font('fonts/Futura.ttf',55)
instructions_font = font.Font('fonts/Futura.ttf',20)
name_font = font.SysFont("rockwell", 40)
score_font = font.SysFont("rockwell", 60)
#colours
black = (0,0,0)
white = (255,255,255)
red = (255,0,0)
orange = (210,70,10)
green = (0,128,0)
navy = (4,8,40)
grey = (100,100,100)
```

Then I created files for the maps (in a folder called ‘maps’).



I have only created 8 so far – enough to be able to make the game possible. I will create more when development is completed. Each map is laid out like this:

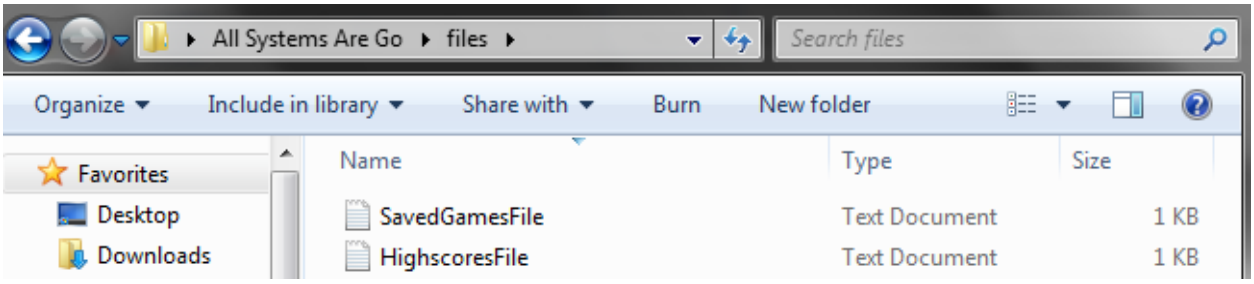


The maximum and minimum power will be different in every level, so their values are stated in the file.

The values in the files represent:

- Self.asteroids = [[x-coordinate, y-coordinate, width, height, density], next asteroid…]
- Self.earth = [x-coordinate, y-coordinate, diameter, radius, radius/diameter]
- Self.goal == [x-coordinate, y-coordinate, diameter, density, density/diameter]

Since I had already made some files, I decided to make my other files (in a folder called ‘files’) now as well.



There is nothing in these two files.

THE PLAYFIELD

Then I started with the playfield. I made a class for everything that will happen in the playfield.

```
class playfield:
#handles all the actual game components
    def __init__(self):
    #create lists from map files
        self.asteroids = []
        self.earth = []
        self.goal = []
        self.rocket = []
```

To read the rest of the files:

```
    def load_map(self,filename):
    #Change global variables to predefined paramaters in the file
        global max_power,min_power
        exec(open(filename,'r').read())
```

Then I created a procedure (still in the 'playfield' class) to use the information from the file to create actual objects.

```
    def create(self,image,properties):
    #scale and blit images onto the playfield, with their densities
        x = properties[0]
        y = properties[1]
        radius = properties[2]
        half_radius = properties[3]
        diameter = properties[2]*2
        scaled_image = transform.scale(image,(diameter,diameter))
        blit_image = screen.blit(scaled_image,(x-radius,y-radius))
        message = density_font.render(str(round(half_radius/radius,2)),1,red)
        w,h = density_font.size(str(round(half_radius/radius,2)))
        message = screen.blit(message,(x-w/2,y-h/2))
```

Working with names such as properties[1] and properties[2] was difficult as I had to remember what each component in each list represented, so I created variables for each component – this is the first 5 lines of the procedure above. The message is the density that is displayed on top of the object on the playfield.

Then I created a procedure that would call the 'create' procedure to draw the earth, asteroids and goal planet (the objects in the map file), and would draw the rest of the objects. Since the 'create' procedure was in a class, I decided to refer to the class as 'game' outside the class to make the coding easier.

Outside the class:

```
game = playfield() #initialise the game
```

Inside the class:

```
    def draw(self):
    #draw all objects (except moon) on the playfield
        global launch_start,power,max_power
        #use properties in file to draw
        game.create(earth_img,self.earth)
        game.create(the_planet,self.goal)
        for i in self.asteroids:
            game.create(asteroid_img,i)
        #only draw rocket if it is launched
        if self.rocket != []:
            x2 = self.rocket[0]-self.rocket[2]/6
            y2 = self.rocket[1]-self.rocket[3]/6
            dx = x2 - self.rocket[0]
            dy = y2 - self.rocket[1]
            rads = atan2(-dy,dx)
            rads %= 2*pi
            degs = degrees(rads) + 90
            rocket = transform.rotate(rocket_img, degs)
            rocket = screen.blit(rocket,(self.rocket[0]-10,self.rocket[1]-10))
```

The moon needs to orbit around Earth, and I didn't know how to calculate the moon's orbit so I researched how. I found a sample of code on http://programarcadegames.com/ which helped me:

```
44      def update(self):
45          """ Update the ball's position. """
46          # Calculate a new x, y
47          self.rect.x = self.radius * math.sin(self.angle) + self.center_x
48          self.rect.y = self.radius * math.cos(self.angle) + self.center_y
49
50          # Increase the angle in prep for the next round.
51          self.angle += self.speed
```

```
for i in range(50):
    # This represents a block
    block = Block(BLACK, 20, 15)

    # Set a random center location for the block to orbit
    block.center_x = random.randrange(SCREEN_WIDTH)
    block.center_y = random.randrange(SCREEN_HEIGHT)
    # Random radius from 10 to 200
    block.radius = random.randrange(10, 200)
    # Random start angle from 0 to 2pi
    block.angle = random.random() * 2 * math.pi
    # radians per frame
    block.speed = 0.008
    # Add the block to the list of objects
    block_list.add(block)
    all_sprites_list.add(block)
```

With the aid of the code above, I created the 'moon' procedure.

```
    def moon(self, moon_visible):
        #draw the moon
        global speed, angle
        #make earth center of orbit
        center_x = self.earth[0] - 10
        center_y = self.earth[1] - 10
        radius = 60
        moon_x = radius * sin(angle) + center_x
        moon_y = radius * cos(angle) + center_y
        #moon will be invisible for the rest of the level if collided with
        if moon_visible: moon = screen.blit(moon_img,(moon_x,moon_y))
        angle += speed
        #to check for collision:
        return moon_x,moon_y
```

To launch the rocket:

```
    def launch(self,x,y,click,angle):
    #use the selected power to give the rocket its initial properties
        global launch_start,power,power_direction,max_power,min_power
        #but only when the user has launched and within the playfield
        if self.rocket == []:
            if (mx < width):
                if not launch_start and click:
                    launch_start = 1
                    power = 10
                elif launch_start and click:
                    power += power_direction
                    if power >= max_power: power_direction = -sqrt(power_direction**2)
                    if power <= min_power: power_direction = sqrt(power_direction**2)
                elif launch_start and not click:
                    self.rocket = [x,y,cos(angle)*power,sin(angle)*power]
                    launch_start = 0
```

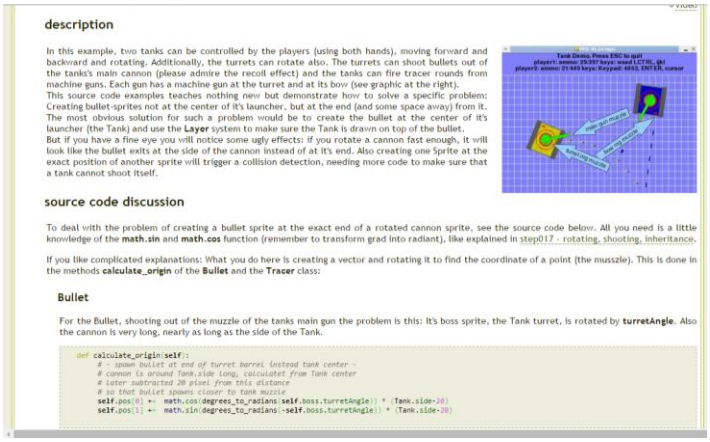
To update the rocket's properties I had to calculate the gravitational effect of the asteroids and the planets on the rocket. I did this in a separate procedure called 'gravity', then called it in an 'update' subroutine.

```
    def gravity(self,i):
    #calculate the rocket's projectile
        distance = sqrt((i[0]-self.rocket[0])**2+(i[1]-self.rocket[1])**2)-i[3]
        angle = atan((i[1]-self.rocket[1])/(i[0]-self.rocket[0]+0.0000001))
        if i[0] < self.rocket[0]: angle += radians(180)
        gravitational_effect = i[3]*(i[2]/(i[2]+distance))**2

        self.rocket[2] += cos(angle)*gravitational_effect/10
        self.rocket[3] += sin(angle)*gravitational_effect/10

    def update(self):
    #Update the rocket's position
        if self.rocket != []:
            for i in self.asteroids:
                game.gravity(i)
            for i in (self.earth,self.goal):
                game.gravity(i)
            self.rocket[0] += self.rocket[2]/10
            self.rocket[1] += self.rocket[3]/10
```

I used the notes from (mostly step 20 of) http://thepythongamebook.com/en:pygame:start to calculate the motion of the rocket.



description

In this example, two tanks can be controlled by the players (using both hands), moving forward and backward and rotating. Additionally, the turrets can rotate also. The turrets can shoot bullets out of the tanks's main cannon (please admire the recoil effect) and the tanks can fire tracer rounds from machine guns. Each gun has a machine gun at the turret and at its bow (see graphic at the right).
This source code examples teaches nothing new but demonstrate how to solve a specific problem: Creating bullet-sprites not at the center of it's launcher, but at the end (and some space away) from it. The most obvious solution for such a problem would be to create the bullet at the center of it's launcher (the Tank) and use the **Layer** system to make sure the Tank is drawn on top of the bullet.
But if you have a fine eye you will notice some ugly effects: if you rotate a cannon fast enough, it will look like the bullet exits at the side of the cannon instead of at it's end. Also creating one Sprite at the exact position of another sprite will trigger a collision detection, needing more code to make sure that a tank cannot shoot itself.

source code discussion

To deal with the problem of creating a bullet sprite at the exact end of a rotated cannon sprite, see the source code below. All you need is a little knowledge of the **math.sin** and **math.cos** function (remember to transform grad into radiant), like explained in step017 - rotating, shooting, inheritance.

If you like complicated explanations: What you do here is creating a vector and rotating it to find the coordinate of a point (the musszle). This is done in the methods **calculate_origin** of the **Bullet** and the **Tracer** class:

Bullet

For the Bullet, shooting out of the muzzle of the tanks main gun the problem is this: It's boss sprite, the Tank turret, is rotated by **turretAngle**. Also the cannon is very long, nearly as long as the side of the Tank.

```
def calculate_origin(self):
    # - spawn bullet at end of turret barrel instead tank center -
    # cannon is around Tank.side long, calculatet from Tank center
    # later subtracted 20 pixel from this distance
    # so that bullet spawns closer to tank muzzle
    self.pos[0] += math.cos(degrees_to_radians(self.boss.turretAngle)) * (Tank.side-20)
    self.pos[1] += math.sin(degrees_to_radians(-self.boss.turretAngle)) * (Tank.side-20)
```

Then I coded the part which checks if the rocket has collided with anything. I decided to add this to the 'update' subroutine, as this is what also affects the rocket's next move. To detect collisions, I used the Pythagorean Theorem.

```
    def update(self,moon_x,moon_y):
    #Update the rocket's position
        if self.rocket != []:
            for i in self.asteroids:
                game.gravity(i)
            for i in (self.earth,self.goal):
                game.gravity(i)
            self.rocket[0] += self.rocket[2]/10
            self.rocket[1] += self.rocket[3]/10
        #check collision with
            #asteroids
            for i in self.asteroids:
                if sqrt((i[0]-self.rocket[0])**2+(i[1]-self.rocket[1])**2) < i[2]:
                    return -1
            #earth
            if sqrt((self.earth[0]-self.rocket[0])**2+\
            (self.earth[1]-self.rocket[1])**2) < self.earth[2]:
                return -1
            #goal planet
            if sqrt((self.goal[0]-self.rocket[0])**2+\
            (self.goal[1]-self.rocket[1])**2) < self.goal[2]:
                return 1
            #screen border
            if self.rocket[0] < 0 or self.rocket[0] > width or\
            self.rocket[1] < 0 or self.rocket[1] > height:
                return -1
            #moon
            if sqrt((moon_x-self.rocket[0])**2+\
            (moon_y -self.rocket[1])**2) < 20:
                return 2
        return 0
```

By doing this I had to change add parameters ‘moon_x’ and ‘moon_y’ to the ‘update’ subroutine.

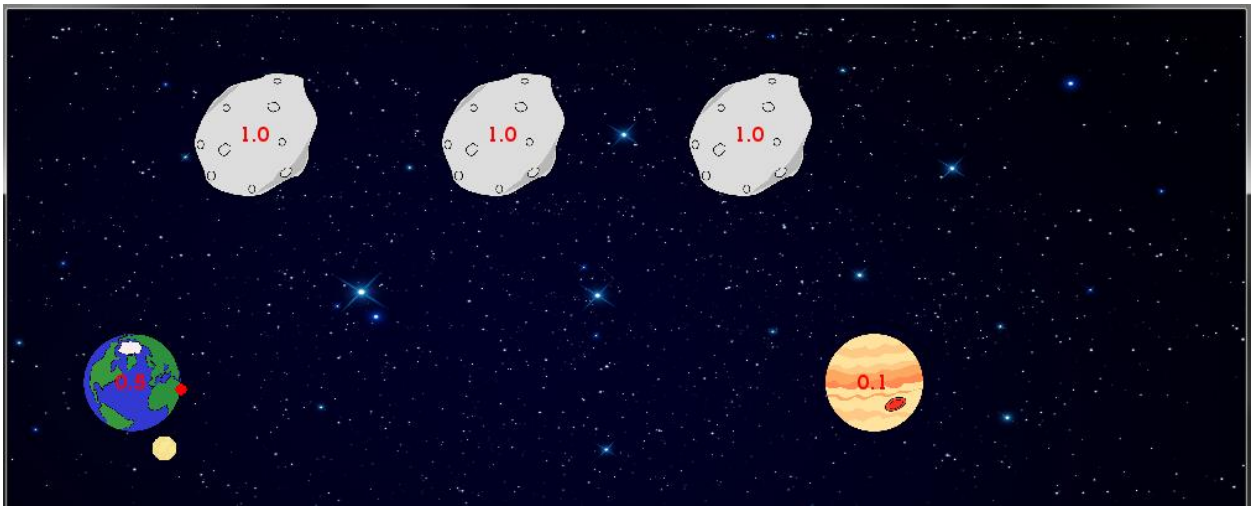
I wanted to test if the actual game part of the program worked before moving on to code the rest of the program. So, outside of the class and in the main part of the code, I wrote this:

```
game = playfield() #initialise the game
mixer.music.play(-1) #play music
#declare and initialise variables
speed = 0.05
angle = pi
launch_start = 0
max_power = 60
min_power = 10
power = 10
power_direction = 2
level = 0
lives = 10
moon_visible = True
#make variable 'maps' so no need so state directory each time
maps = ['maps/map'+str(i+1)+'.txt' for i in range(10)]
game.load_map(maps[level]) #load first level
```

```
cont = True
while cont:
    ###main program loop starts here###
    mx,my = mouse.get_pos() #mx and my are the coordinates of the cursor
    lc = mouse.get_pressed()[0] #lc = left click
    screen.blit(background_img,(0,0))
    game.draw() #call 'draw' module
    moon_x,moon_y = game.moon(moon_visible) #return moon coordinates
    #find the firing angle based on cursor position
    f_angle = atan((my-game.earth[1])/(mx-game.earth[0]+0.00000001))
    if mx < game.earth[0]: f_angle += radians(180)
    x = cos(f_angle)*game.earth[2]+game.earth[0] #and hence calulate the
    y = sin(f_angle)*game.earth[2]+game.earth[1] #rocket's next poition
    #launch rocket
    game.launch(x,y,lc,f_angle)
    if game.rocket == []: #if rocket not launched yet, draw red dot
        draw.circle(screen,red,(round(x),round(y)),5)
```

```
    action = game.update(moon_x,moon_y)
    #Will return -1 if crashed, 1 if succeed, 2 if hit moon
    if action < 0:
        #reset level and report crash
        lives -=1
        print(lives)
        game.rocket = []
        power = min_power
    elif action == 2:
        #make moon invisible if it was visible before
        if moon_visible:
            moon_sound.play()
            moon_visible = False
            lives +=1
            print(lives)
    elif action == 1:
        #go to next level and reset rocket and moon
        cheer_sound.play()
        level +=1
        print(level)
        game.load_map(maps[level]) #load the next map
        game.rocket = []
        power = min_power
        moon_visible = True
    display.flip()
    time.delay(30)
```

Then, when I ran the program, all the objects were positioned in the right place according to map1. However, the red dot didn't move as I moved the cursor.



This showed that the main loop did run normally (the moon orbited the Earth) and the file was read properly and that the drawing modules in the 'playfield' class worked. However, since the red dot didn't move, something was wrong with obtaining the cursor position. To check this, I wrote this:

```
mx,my = mouse.get_pos() #mx and my are the coordinates of the cursor
print(mx,my) # if same each time, proble is in this loop
```

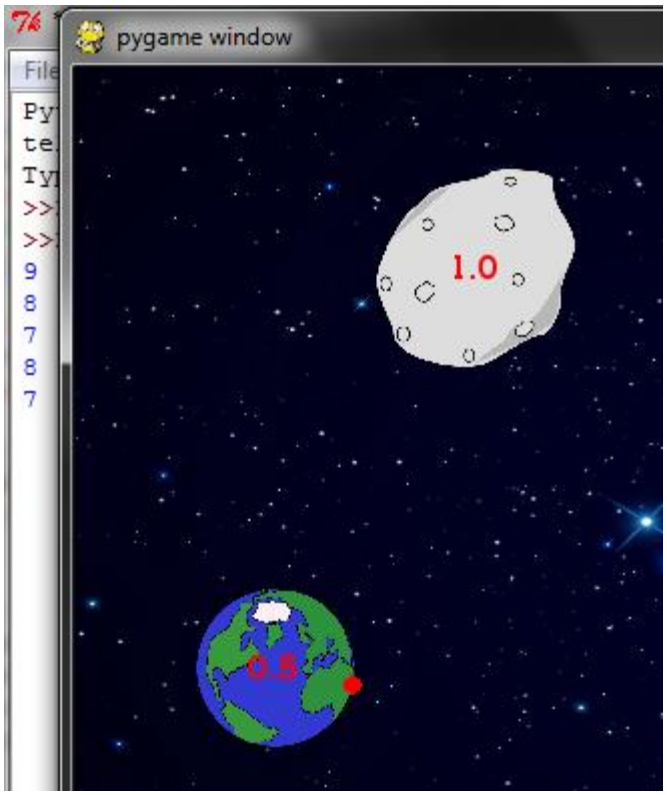
When I ran the program, the same coordinates were printed. So I knew that the problem was in the main loop. So I removed the `print(mx,my)` line and tried to diagnose the problem using a trace table. In doing so, I realised that there was no part of the code which handled events, and since moving the mouse was an event, it wouldn't pick it up. So I added this to the start of the loop:

```
#loop for events (will expand)
for evnt in event.get():
    if evnt.type == QUIT:
        cont = False #exit loop
```

Things to note:

- I did not need a construct for mouse movement here – I just needed an events handler
- Since 'continue' is a keyword in Python, my variable had to be called 'cont'

Then I ran the program again.

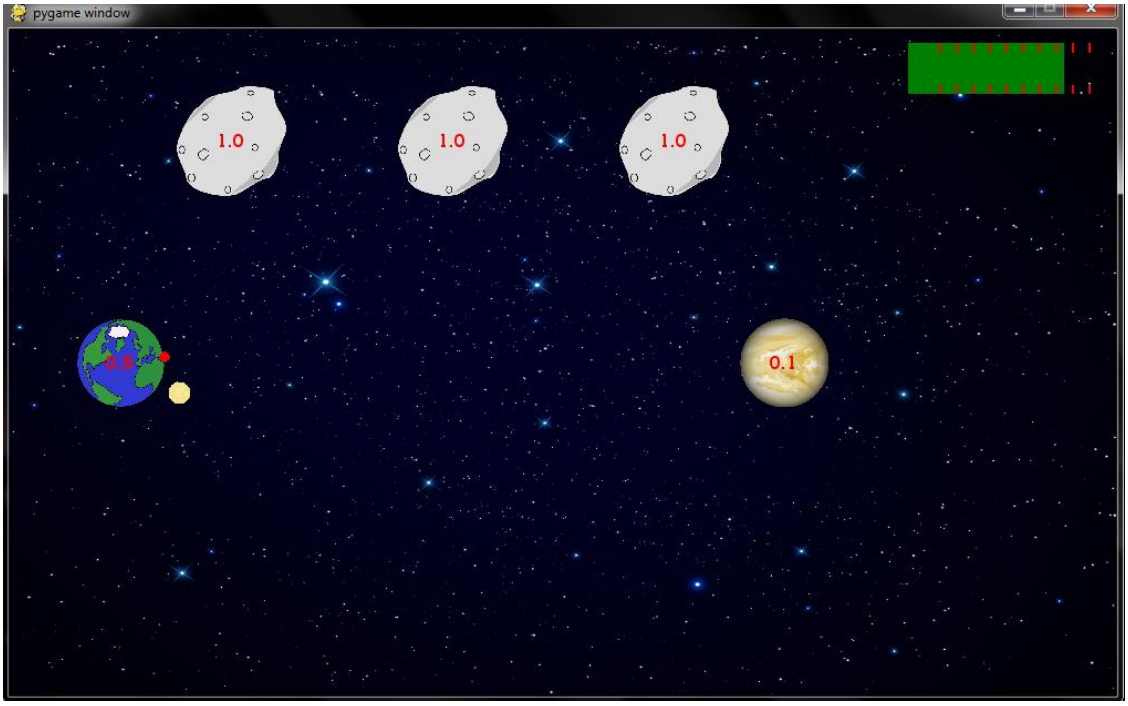


This time the red dot did move. Therefore I was able to test the rocket launch and flight. Once I launched the rocket, the red dot disappeared and changed to a rocket, as planned, and the rocket did fly and curve towards the objects, as planned. I crashed the rocket into the asteroid tree times, then crashed into the moon, then the screen border on the fourth flight. On the far left of the image above shows the number of lives that that was printed, which is as expected. So now I know that all the modules in the 'playfield' class works!

However, I did not know my launch power as I did not draw a power bar. So, I wrote this:

```
    #drawing the power bar
    draw.rect(screen,green,(812,12,power*(180/max_power)-3,46))
    for i in range(840,990,15):
        #red lines are fiducial markers
        draw.line(screen,red,(i,12),(i,20),2)
        draw.line(screen,red,(i,50),(i,57),2)
```

I ran the program:



The power bar moved as I held down the lift click.

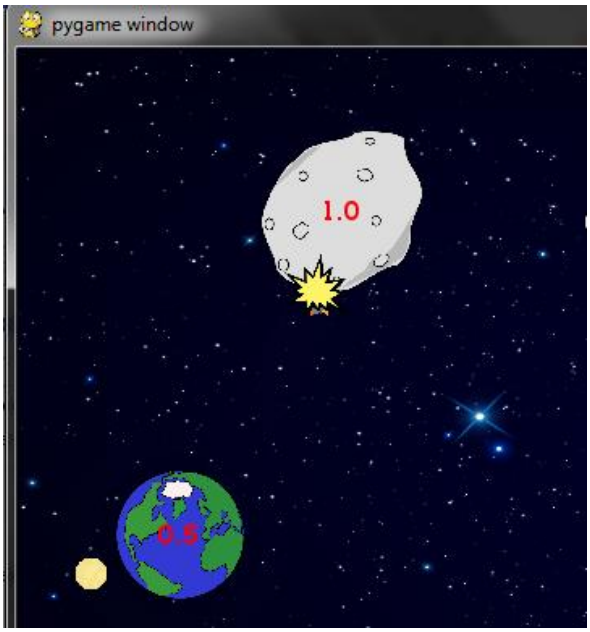
Then I added a crashing effect. In the 'playfield' class:

```
    def crash(self):
    #play crash sound and change rocket image to exploded rocket image
        crash_sound.play()
        screen.blit(explosion_img,(self.rocket[0]-20,self.rocket[1]-20))
        display.flip()
```

In the main loop I added `game.crash()`:

```
    if action < 0:
        #reset level and report crash
        game.crash()
        lives -=1
        print(lives)
        game.rocket = []
        power = min_power
```

When I ran the program and crashed, the sound played on top of the music and this was displayed:



Also, I now need to display the number of lives and the level in the same window. I removed `print(lives)` and `print(level)`.

Since the panel with this information has buttons on it too, as well as on all the pages in the game, I thought it would be better to create another class just for making buttons. So I wrote this:

```
class shapes(sprite.Sprite):
#class for buttons
    def __init__(self,x,y,width,height,border,option):
        #define a button
        sprite.Sprite.__init__(self)
        self.box = Surface([x+width,y+height],border)
        if option == 1: #1 for rectange, 2 for circle
            #draw button and outline
            draw.rect(screen,black,[x,y,width,height],0)
            draw.rect(screen,white,[x,y,width,height],border)
        if option == 2:
            radius = int(width/2)
            draw.ellipse(screen,black,[x,y,width,height],0)
            for i in range(300): #loop for creating circle outline
                ang = i * 3.14159 * 2 / 300
                dx = int(cos(ang) * radius)
                dy = int(sin(ang) * radius)
                bx = x + dx
                by = y + dy
                draw.circle(screen,white,[bx+radius,by+radius],(border//3))
        self.rect = self.box.get_rect()
        self.rect.x = x #get the coordinates of the button
        self.rect.y = y #to check for collisions (mouse clicks)
```

I called the class, ‘shapes’ because it will be used to make shapes that won’t be used as buttons as well. Also, by using ‘option’, I didn’t need to create two separate modules for the two shapes. Instead it would be another parameter. Also, the reason for why the border for the circular buttons aren’t drawn like the rectangular ones is because if it was, the border would look grainy like:

Built-in circle outlines: http://www.nerdparadise.com/tech/python/pygame/basics/part4/

The code for the smooth circle border was also taken from http://www.nerdparadise.com/tech/python/pygame/basics/part4/

Then, also because of repeats, I created procedures for the text for the home button and sound button:

```
def homebutton(x,y):
    #create text for home button with given coordinates
    screen.blit(button_font_14.render("Home",1,green),(x+20,y+30))

def soundbutton(x,y,colour,option):
    #create text for sound button with given coordinates,
    #colour and whether the text is 'on' or 'off
    screen.blit(button_font_14.render("Sound",1,colour),(x+15,y+25))
    screen.blit(button_font_14.render(option,1,colour),(x+27,y+37))
```

I also made a subroutine that will be called when the sound button was clicked – to change its label:

```
def sounds(option):
#change text on button when
#user turns sound on/off
    if option == "on":
        mixer.music.unpause()
        option = "off"
    else:
        mixer.music.pause()
        option = "on"
    return option
```

Initially I made the title an image and was going to blit it like the other images onto the screen. However, because it was text, the edges were obviously fuzzy, even if only a bit. So I decided to blit the actual text on the screen. Since the text would be positioned in the same place relative to each word, I made a procedure for the title:

```
def title(x,y,page):
#draw the title, 'All Systems Are Go'
    crimson = (180,0,0)
    if page != "home" and page !="nameinput" and page !="loadgame":
        #only these three pages have a box surrounding the title
        shapes(x-6,y+20,180,260,3,1)
    #the text
    screen.blit(font.SysFont("rockwell", 130).render("All",1,crimson),(x,y))
    screen.blit(font.SysFont("rockwell", 46).render("Systems",1,crimson),(x,y+120))
    screen.blit(font.SysFont("rockwell", 46).render("Are",1,crimson),(x+95,y+158))
    screen.blit(font.SysFont("rockwell", 130).render("Go",1,crimson),(x-8,y+145))
```

Creating the panel on the playfield page:

```
draw.rect(screen,grey,[width,0,200,height])#grey panel
reset_button = shapes(815,425,80,80,3,2)#buttons
save_button = shapes(905,425,80,80,3,2)
home_button = shapes(815,515,80,80,3,2)
sound_button = shapes(905,515,80,80,3,2)
title(816,60,page)
shapes(810,10,180,50,3,1)#these two won't be buttons
shapes(810,355,180,60,3,1)
#text
screen.blit(density_font.render("Level " + str(level+1),1,green),(815,360))
screen.blit(density_font.render("Lives remaining: " + str(lives),1,green),(815,385))
screen.blit(button_font_14.render("Reset",1,red),(836,450))
screen.blit(button_font_14.render("rocket",1,red),(833,462))
screen.blit(button_font_14.render("Save and",1,green),(912,450))
screen.blit(button_font_14.render("exit",1,green),(930,462))
homebutton(815,515)
soundbutton(905,515,red,change_to)
#drawing the power bar
draw.rect(screen,green,(812,12,power*(180/max_power)-3,46))
for i in range(840,990,15):
    #red lines are fiducial markers
    draw.line(screen,red,(i,12),(i,20),2)
    draw.line(screen,red,(i,50),(i,57),2)
```

I added the code above to the main program loop, then ran the program. This is what was displayed:

The buttons are still ineffective but the level and lives show correctly. However, I found that the code for the panel was too bulky to put in the main loop so I most of it (and the code for the power bar) in a module called ‘game_panel’. I left out the drawing of the actual buttons from game_panel.

So now I have this:

```
def game_panel():
    draw.rect(screen,grey,[width,0,200,height])#grey panel
    reset_button = shapes(815,425,80,80,3,2)#buttons
    save_button = shapes(905,425,80,80,3,2)
    home_button = shapes(815,515,80,80,3,2)
    sound_button = shapes(905,515,80,80,3,2)
    title(816,60,page)
    shapes(810,10,180,50,3,1)#these two won't be buttons
    shapes(810,355,180,60,3,1)
    #text
    screen.blit(density_font.render("Level " + str(level+1),1,green),(815,360))
    screen.blit(density_font.render("Lives remaining: " + str(lives),1,green),(815,385))
    screen.blit(button_font_14.render("Reset",1,red),(836,450))
    screen.blit(button_font_14.render("rocket",1,red),(833,462))
    screen.blit(button_font_14.render("Save and",1,green),(912,450))
    screen.blit(button_font_14.render("exit",1,green),(930,462))
    homebutton(815,515)
    soundbutton(905,515,red,change_to)
    #drawing the power bar
    draw.rect(screen,green,(812,12,power*(180/max_power)-3,46))
    for i in range(840,990,15):
        #red lines are fiducial markers
        draw.line(screen,red,(i,12),(i,20),2)
        draw.line(screen,red,(i,50),(i,57),2)
```

And this in the main loop:

```
    screen.blit(background_img,(0,0))
    draw.rect(screen,grey,[width,0,200,height])
    reset_button = shapes(815,425,80,80,3,2)
    save_button = shapes(905,425,80,80,3,2)
    home_button = shapes(815,515,80,80,3,2)
    sound_button = shapes(905,515,80,80,3,2)
    game_panel()
    game.draw() #call 'draw' module
```

Now I had to write the code for when the game ends. The game can end when:

- The player has ran out of lives, or
- The player has completed all the levels

So I wrote a module in the ‘playfield’ class:

```
    def check_gameover(self,lives,page,endgame,complete):
    #check if game has ended and display appropriate message if so
        if lives < 0 or complete:
            if lives < 0: #if out of lives
                death_sound.play()
                screen.blit(button_font_40.render("GAME OVER",1,red),(280,290))
            if complete: #if out of levels
                screen.blit(button_font_40.render("GAME COMPLETE",1,red),(240,290))
            screen.blit(button_font_14.render("Click to continue",1,red),(350,330))
            page = "nameinput"
            endgame = True #endgame is used to differenciate
            cont = True    #between saving a complete and incomplete game
            while cont:
            #only go to "nameinput" once the user has clicked the screen
                display.flip()
                for evnt in event.get():
                    if evnt.type == MOUSEBUTTONUP:
                        cont = False
        return page, endgame
```

Since I used two new variables, ‘endgame’ and ‘complete’, I need to add them to the main program. In the part where I declared and initialised the other variables, I added:

```
complete = False
endgame = False
```

And in the main loop, I changed the IF function for completing the level to:

```
    elif action == 1:
        #go to next level and reset rocket and moon
        cheer_sound.play()
        level += 1
        try: game.load_map(maps[level]) #try to load the next map
        except: #but if there are no maps left:
            complete = True
            endgame = True
            page,endgame = game.check_gameover(lives,page,endgame,complete)
        game.rocket = []
        power = min_power
        moon_visible = True
        complete = False

    #check if game has ended
    page,endgame = game.check_gameover(lives,page,endgame,complete)
    display.flip()
    time.delay(30)
```

Even though I haven’t made the other pages, such as “nameinput” yet, I can test if the end of game displays worked. So I temporarily did:

```
level = 7
lives = 0
```
in the initialisation bit, then ran the program twice. The first time, I crashed. The ‘death crash’ sound was played and this is what was displayed:

Then I restarted the program and guided the rocket to the goal planet. This is what was displayed:

Since both displays are correct but an error is displayed when I click the screen since there is no 'nameinput' page yet, now I will expand the main loop so that there are multiple pages.

In the main loop is one large If…Elif…Else statement (as demonstrated in the pseudocode). When I wrote it, I commented out the statements for the pages I haven't started working on yet, so that the program can still run without errors. All the code that was originally in the main loop is now in the playfield part, so now the main loop is thus:

```
cont = True
while cont:
    ###main program loop starts here###
    screen.fill(grey) #since most pages are grey
    #handle events
    for evnt in event.get():
        if evnt.type == QUIT:
            cont = False #exit loop

    if page == "home":
        #variables reset for every new game
        level = 0
        lives = 10
        name = ""
        game.load_map(maps[level]) #load first level
        moon_visible = True
        complete = False
        endgame = False
```

```
    #elif page == "loadgame":

    #elif page == "selectgame":

    elif page == "playfield":
        mx,my = mouse.get_pos() #mx and my are the coordinates of the cursor
        lc = mouse.get_pressed()[0] #lc = left click
        screen.blit(background_img,(0,0))
        draw.rect(screen,grey,[width,0,200,height])
        reset_button = shapes(815,425,80,80,3,2)
        save_button = shapes(905,425,80,80,3,2)
        home_button = shapes(815,515,80,80,3,2)
        sound_button = shapes(905,515,80,80,3,2)
        game_panel()
        game.draw() #call 'draw' module
        moon_x,moon_y = game.moon(moon_visible) #return moon coordinates
        #find the firing angle based on cursor position
        f_angle = atan((my-game.earth[1])/(mx-game.earth[0]+0.00000001))
        if mx < game.earth[0]: f_angle += radians(180)
        x = cos(f_angle)*game.earth[2]+game.earth[0] #and hence calulate the
        y = sin(f_angle)*game.earth[2]+game.earth[1] #rocket's next poition
        #launch rocket
        game.launch(x,y,lc,f_angle)
```

```
        if game.rocket == []: #if rocket not launched yet, draw red dot
            draw.circle(screen,red,(round(x),round(y)),5)
        action = game.update(moon_x,moon_y)
        #Will return -1 if crashed, 1 if succeed, 2 if hit moon
        if action < 0:
            #reset level and report crash
            game.crash()
            lives -=1
            game.rocket = []
            power = min_power
        elif action == 2:
            #make moon invisible if it was visible before
            if moon_visible:
                moon_sound.play()
                moon_visible = False
        elif action == 1:
            #go to next level and reset rocket and moon
            cheer_sound.play()
            level += 1
            try: game.load_map(maps[level]) #try to load the next map
            except: #but if there are no maps left:

                complete = True
                endgame = True
                page,endgame = game.check_gameover(lives,page,endgame,complete)
            game.rocket = []
            power = min_power
            moon_visible = True
            complete = False
        #check if game has ended
        page,endgame = game.check_gameover(lives,page,endgame,complete)

    #elif page == "instructions":

    #elif page == "about":

    #elif page == "highscores":

    #elif page == "nameinput":

    display.flip()
    time.delay(20)
quit()#quit when cont = False
```

I also added some variables in the initialisation part:

```
change_to = "off"
page = "home"
```

… And moved some of the variables (the ones that need to be reset for every new game) to the homepage section.

Finally, I had to make the buttons on the game panel work. So, in the events for-loop, I added:

```
if evnt.type == MOUSEBUTTONUP:#if the user clicks something
    pos = mouse.get_pos()
    if (page == "home" or page == "playfield") and sound_button.rect.collidepoint(pos):
        change_to = sounds(change_to)
    elif home_button.rect.collidepoint(pos):
        page = "home"
    else:
        if page == "playfield": #buttons on the playfield page
            if reset_button.rect.collidepoint(pos):
                game.rocket = []
                power = min_power
                lives -=1
            if save_button.rect.collidepoint(pos):
                page = "nameinput"
```

Since the sound button and home button will be on multiple pages, I added the If statement for those at the start of the events loop. This loop basically states that if the user does anything in the game, get the mouse position, see what the mouse position corresponds to, then carry out the appropriate action (in most cases, change the page).

When I clicked on the 'Sound Off' button, the music stopped and the text changed to 'Sound On'. When I clicked the home button or the save button, a blank screen appeared (because the pages haven't been created yet). When I clicked on the reset button, the number of lives decremented by 1.

THE 'HOME' PAGE

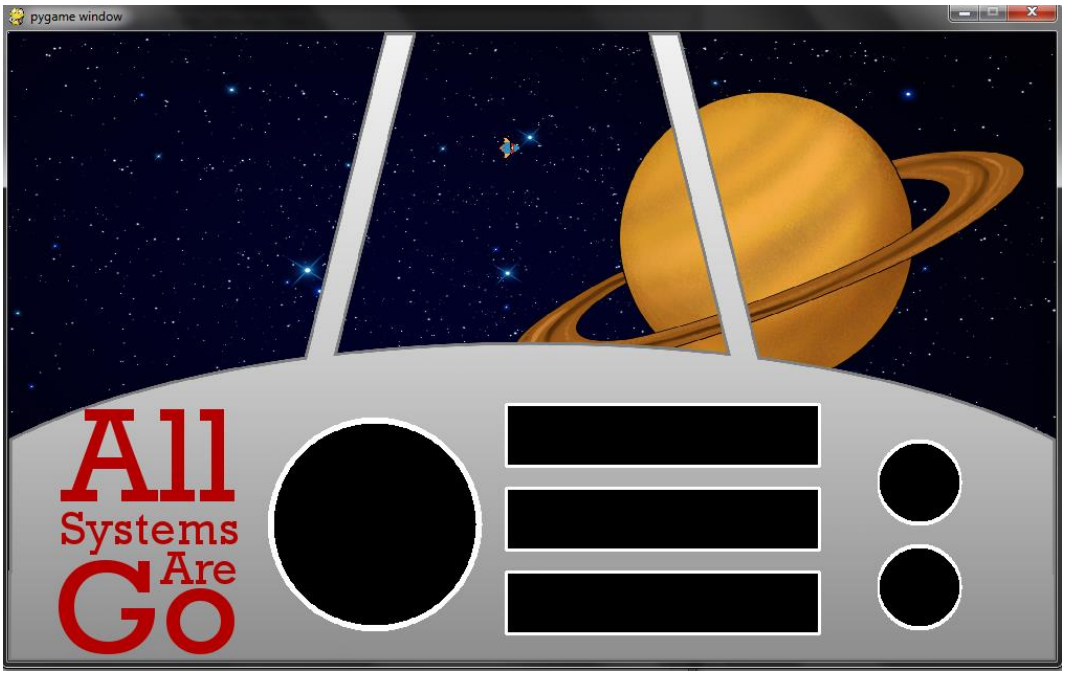
After the variables, I started making the visuals of the homepage. I added:

```
        screen.blit(background_img,(0,0))#sky image

        #making the rocket orbit
        rocket = transform.rotate(rocket_img,-90)
        center_x, center_y = 500,800 #center of orbit
        radius = 700 #radius of orbit
        speed = 0.02
        x = radius * sin(angle) + center_x
        y = radius * cos(angle) + center_y
        screen.blit(rocket,(x,y))
        angle -= speed

        screen.blit(home_img,(0,0))#ship image
        title(50,325,page)
        #buttons
        start_button = shapes(250,370,200,200,10,2)
        instructions_button = shapes(475,355,300,60,3,1)
        highscores_button = shapes(475,435,300,60,3,1)
        exit_button = shapes(475,515,300,60,3,1)
        about_button = shapes(830,390,80,80,3,2)
        sound_button = shapes(830,490,80,80,3,2)
        home_button = shapes(0,0,0,0,0,1)
```

The code for the orbiting rocket is the same as for the moon on the playfield. When I ran the program, this was displayed with the music:



Before adding the text to the buttons, I made the rotating red markers on the 'Play' button. Pygame has a built in rotation function, but it rotates around the image corner. I wanted a rotation around the centre of the image, so I created a procedure specifically for rotating images:

```
def rotate(image, rect, angle,x,y):
#rotate an image around a given point
    rot_image = transform.rotate(image, angle)
    rot_rect = rot_image.get_rect(center=rect.center)
    screen.blit(rot_image,(rot_rect.x+x, rot_rect.y + y))
```

Then, in the homepage section of the main loop, I wrote:

```
        #drawing the rotating red dial
        rotation -= 2 #degrees
        rotate(dial_img,dial_img.get_rect(),rotation,252,372)
        rotate(dial_img,dial_img.get_rect(),rotation-22.5,252,372)
        #text
        screen.blit(button_font_55.render("Play",1,green),(290,435))
        screen.blit(button_font_40.render("Instructions",1,green),(510,360))
        screen.blit(button_font_40.render("Highscores",1,green),(515,440))
        screen.blit(button_font_40.render("Exit",1,green),(590,520))
        screen.blit(button_font_14.render("About",1,green),(845,420))
        soundbutton(830,490,green,change_to)
```

When I ran the program an error occurred. I hadn't defined 'rotation'. So, outside of the loop in the declaring and initialising variables section, I added `rotation = 0`. Then I ran the program again:

The rotation procedure works.

In the events for-loop, I added:

```
elif page == "home": #buttons on the homepage
    if start_button.rect.collidepoint(pos):
        page = "loadgame"
    if instructions_button.rect.collidepoint(pos):
        page = "instructions"
    if highscores_button.rect.collidepoint(pos):
        page = "highscores"
    if exit_button.rect.collidepoint(pos):
        cont = 0
    if about_button.rect.collidepoint(pos):
        page = "about"
```

When I clicked on the 'Sound Off' button, the music stopped and the text changed to 'Sound On'.

When I clicked on the other buttons on the homepage, a blank grey screen appeared. This shows that the page does change, but since I have not coded the other pages yet, only the background is filled.

THE 'LOADGAME' PAGE

In the "loadgame" section of the main program loop, I added this:

```
elif page == "loadgame":
    #display background images and title
    screen.blit(background_img,(0,0))
    screen.blit(home_img,(0,0))
    title(50,325,page)
    #call the 'shapes' class to create buttons
    new_button = shapes(280,370,200,200,10,2)
    load_button = shapes(520,370,200,200,10,2)
    home_button = shapes(900,500,80,80,3,2)
    homebutton(900,500) #text for the home button
    #drawing the rotating red dials
    rotation +=2
    rotate(dial_img,dial_img.get_rect(),rotation,282,372) #rotate anticlockwise
    rotate(dial_img,dial_img.get_rect(),-rotation,522,372) #rotate clockwise (-rotation)
    #text
    screen.blit(button_font_40.render("New",1,green),(335,425))
    screen.blit(button_font_40.render("game",1,green),(322,455))
    screen.blit(button_font_40.render("Load",1,green),(568,425))
    screen.blit(button_font_40.render("game",1,green),(562,455))
```

Then when I ran the program and clicked on the 'Play' button on the home page, this was displayed:

Clicking the 'Home' button brought me back to the home page, but clicking the two other buttons did nothing. So, I added this to the end of the events loop in the MOUSEBUTTONUP section:

```
            elif page == "loadgame": #buttons on the load page
                if new_button.rect.collidepoint(pos):
                    page = "playfield"
                if load_button.rect.collidepoint(pos):
                    page = "selectgame"
```

Then I ran the program again. When I clicked on the 'New game' button, the playfield page is displayed, with level = 1 and lives remaining = 10. When I clicked on the 'Load game', a black grey screen appeared. So the next page to program is the 'selectgame' page.

THE 'SELECTGAME' PAGE

In the "selectgame" section of the main program loop, I added this:

```
    elif page == "selectgame":
        #title and text and box for heading
        title(806,0,page)
        shapes(15,15,300,60,3,1)
        screen.blit(button_font_40.render("Load Game",1,green),(50,20))
        #homebutton and text for homebutton
        home_button = shapes(900,500,80,80,3,2)
        homebutton(900,500)
```

When I ran the program and clicked on 'Play' (home page), then 'Load game' (loadpage), this was displayed:

```
#list of saved games, detected by the row in the table (the rectangle they're in)
savedgames = [shapes(15,210,475,60,3,1), shapes(15,310,475,60,3,1), shapes(15,410,475,60,3,1)]
```

Next I wanted the table for the list of saved games. However, the different rows in the table had to be identifiable when clicked on. So, I added this to the end of the code before:

By making savedgames a list, I could use iteration to find which saved game had been clicked (savedgames[0], savedgames[1] of savedgames[2]).

This was displayed:

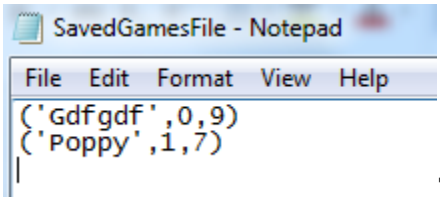
Then I had to obtain the saved games from the file and save them to an array in the game – games_lst.

```
#open and read saved games form SavedGamesFile in a array format. Then close file
with open('files/SavedGamesFile.txt', 'r', encoding='utf-8') as f:
    games_lst = [literal_eval(line.strip()) for line in f]
f.close()
# to only display the 3 newest saved games:
while len(games_lst) >  3:
    del games_lst[0]
```

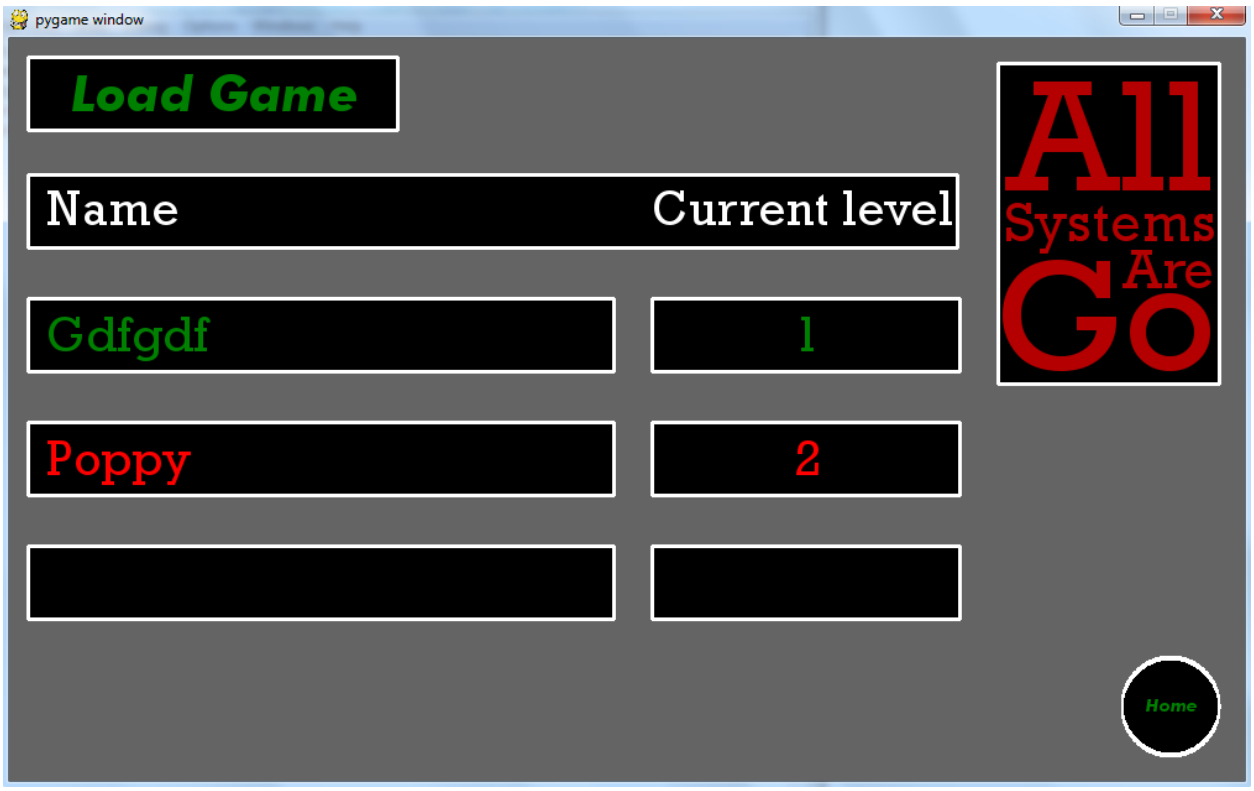
To display the saved games on the screen, I wrote this:

```
#draw box and text for table heading
shapes(15,110,753,60,3,1)
screen.blit(name_font.render("Name",1,white),(30,113))
screen.blit(name_font.render("Current level",1,white),(520,113))
#draw boxes for the second column of the table
for i in range(3):
    shapes(520,110+(i+1)*100,250,60,3,1)
# display text of all the names and levels in games_lst
for j in range(len(games_lst)):
    if j%2==1: #change colour of text between red and green
        colour = red
    else:
        colour = green
    screen.blit(name_font.render(games_lst[j][0],1,colour),(30,115+(j+1)*100))
    screen.blit(name_font.render(str(games_lst[j][1]+1),1,colour),(635,115+(j+1)*100))
```

Then I ran the program. The table was displayed, but there were no entries since SAVEDGAMESFILE was empty. So I added some entries in the file to text if my code worked or not.



Then when I ran the program again this was the display:



So the code works (as I made level = 0 for the first level). The code to display the names in the table was too bulky for the main program loop, so I put it in a procedure called ‘currentgames’, then added a line to call it in the main loop:

```
def currentgames(games_lst):
    #draw box and text for table heading
    shapes(15,110,753,60,3,1)
    screen.blit(name_font.render("Name",1,white),(30,113))
    screen.blit(name_font.render("Current level",1,white),(520,113))
    #draw boxes for the second column of the table
    for i in range(3):
        shapes(520,110+(i+1)*100,250,60,3,1)
    # display text of all the names and levels in games_lst
    for j in range(len(games_lst)):
        if j%2==1: #change colour of text between red and green
            colour = red
        else:
            colour = green
        screen.blit(name_font.render(games_lst[j][0],1,colour),(30,115+(j+1)*100))
        screen.blit(name_font.render(str(games_lst[j][1]+1),1,colour),(635,115+(j+1)*100))
```

…

```
        currentgames(games_lst)
```

However, nothing happened when I clicked on the names, so I added this in the event loop:

```
                elif page == "selectgame": #saved games on the select game page
                    for i in range(len(games_lst)):
                        if savedgames[i].rect.collidepoint(pos):
                            name = games_lst[i][0]
                            level = games_lst[i][1]
                            lives = games_lst[i][2]
                            game.load_map(maps[level])
                            page = "playfield"
```

In this code, the collision of all the boxes holding the names of the saved games will be checked. If the box being the checked matches the position of mouse click, the name, level and lives remaining will change from the default to those of the game selected. Then the right map will load and the page changed.

I ran the program, then clicked on Poppy’s game (one I made for texting). This is what was displayed:

The level and lives match the file, so the code works.

THE 'INSTRUCTIONS' PAGE

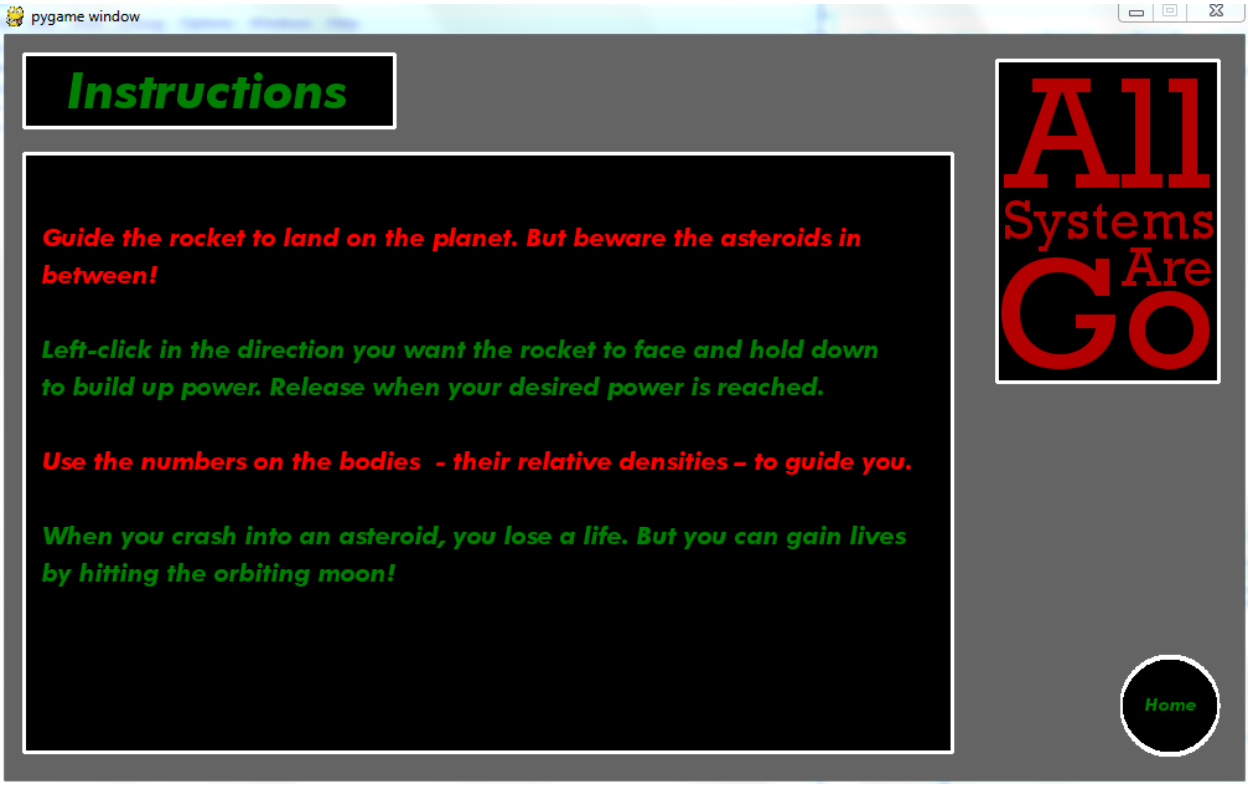
In the "instructions" section of the main program loop, I added this:

```
    elif page == "instructions":
        #title and text and box for heading
        title(806,0,page)
        shapes(15,15,300,60,3,1)
        screen.blit(button_font_40.render("Instructions",1,green),(50,20))
        #home button and its text
        home_button = shapes(900,500,80,80,3,2)
        homebutton(900,500)
        #call the module with the instrctions text
        instructions()
```

```
def instructions():
    #box to store instructions
    shapes(15,95,750,483,3,1)
    #instructions text
    i = 150
    screen.blit(instructions_font.render("Guide the rocket to land on the planet. But beware the asteroids in",1,red),(30,i))
    screen.blit(instructions_font.render("between!",1,red),(30,i+30))
    screen.blit(instructions_font.render("Left-click in the direction you want the rocket to face and hold down ",1,green),(30,i+90))
    screen.blit(instructions_font.render("to build up power. Release when your desired power is reached.",1,green),(30,i+120))
    screen.blit(instructions_font.render("Use the numbers on the bodies  - their relative densities - to guide you.",1,red),(30,i+180))
    screen.blit(instructions_font.render("When you crash into an asteroid, you lose a life. But you can gain lives",1,green),(30,i+240))
    screen.blit(instructions_font.render("by hitting the orbiting moon!",1,green),(30,i+270))
```

I added an ‘instructions’ module to make the main program loop less cluttered.

When I ran the program and went on the instructions page, this was displayed:



THE ‘ABOUT’ PAGE

In the “about” section of the main program loop, I added this:

```
    elif page == "about":
        #title and text and box for heading
        title(806,0,page)
        shapes(15,15,300,60,3,1)
        screen.blit(button_font_40.render("About",1,green),(100,20))
        #home button and its text
        home_button = shapes(900,500,80,80,3,2)
        homebutton(900,500)
        #call the module with the 'about' text
        about()
```

```
def about():
    #box to hold text
    shapes(15,95,750,483,3,1)
    #the text
    x = 50
    i = 150
    screen.blit(instructions_font.render("Giny Huynh 2015",1,white),(300,i))
    screen.blit(instructions_font.render("Images:",1,green),(x,i+60))
    screen.blit(instructions_font.render("clipartlord.com",1,red),(x,i+90))
    screen.blit(instructions_font.render("clker.com",1,red),(x,i+120))
    screen.blit(instructions_font.render("mycutegraphics.com",1,red),(x,i+150))
    screen.blit(instructions_font.render("Music:",1,green),(x,i+210))
    screen.blit(instructions_font.render('"Pamgaea" Kevin MacLeod (incompetech.com) ',1,red),(x,i+240))
    screen.blit(instructions_font.render("Licensed under Creative Commons: By Attribution 3.0 ",1,red),(x,i+270))
    screen.blit(instructions_font.render("http://creativecommons.org/licenses/by/3.0/",1,red),(x,i+300))
```

For the same reason as with the instructions page, I added an ‘about’ module to make the main program loop less cluttered.

When I ran the program and went on the about page, this was displayed:

THE ‘HIGHSCORES’ PAGE

In the “highscores” section of the main program loop, I added this:

```
    elif page == "highscores":
        #title and text and box for heading
        title(806,0,page)
        shapes(15,15,300,60,3,1)
        screen.blit(button_font_40.render("Highscores",1,green),(50,20))
        #home button and its text
        home_button = shapes(900,500,80,80,3,2)
        homebutton(900,500)
        #clear button and its text
        clear_button = shapes(800,500,80,80,3,2)
        screen.blit(button_font_14.render("Clear",1,red),(820,520))
        screen.blit(button_font_14.render("highscores",1,red),(804,535))
```

When I ran the program, this was displayed:

Clicking the 'Clear highscore's button did nothing, so in the events loop, I added this:

```
                elif page == "highscores": #clear button on high scores page
                    if clear_button.rect.collidepoint(pos):
                        with open("files/HighscoresFile.txt", "w") as f:
                            f.close()
```

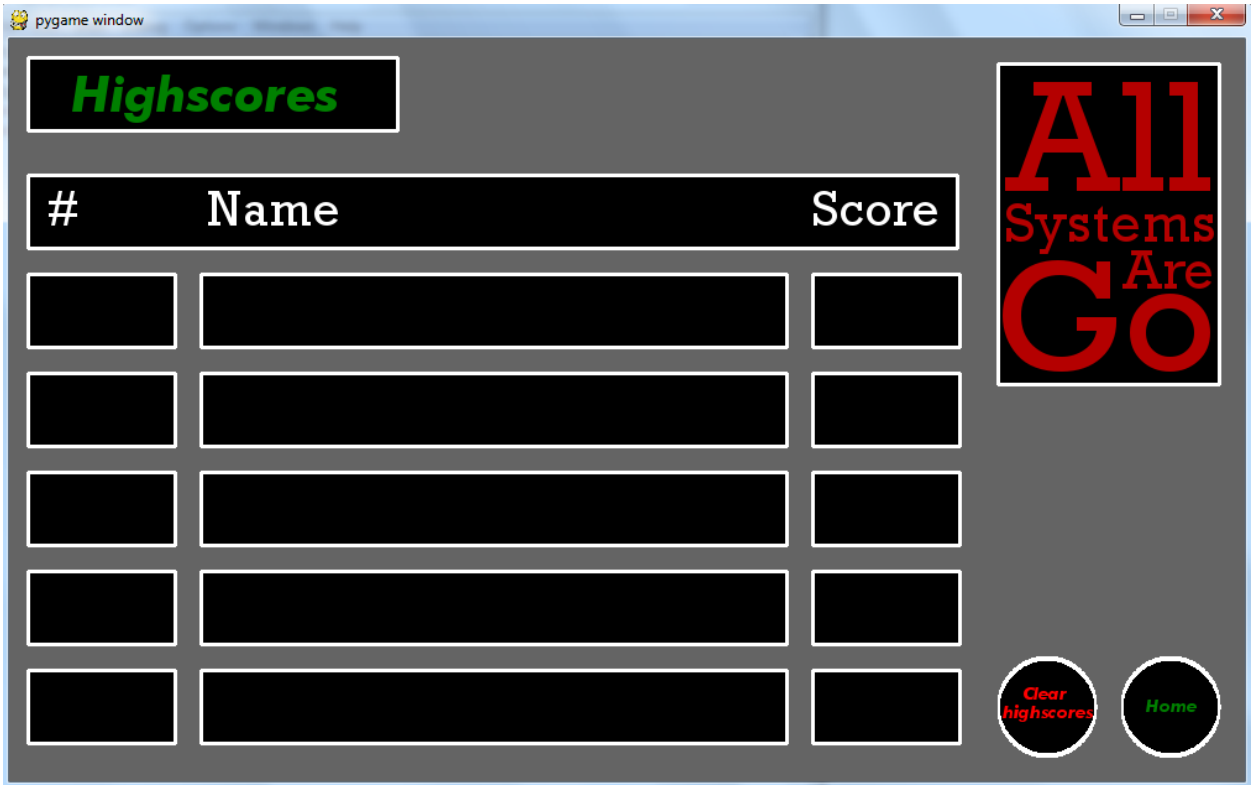
Opening a text file in 'write' mode clears all the contents of the file before anything is written in it. I have used this feature to clear the high scores file.

To make the highscores table, I had to first store the contents of HighscoresFile to an array, which I called scores. Then I made a subroutine called ‘scorestable’ to display the items in ‘scores’.

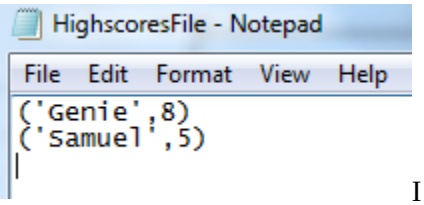
```
        #store contents of HighscoresFile into array
        with open('files/HighscoresFile.txt', 'r', encoding='utf-8') as f:
            scores = [literal_eval(line.strip()) for line in f]
        f.close()
        #call the module that makes the score table
        scoretable(scores)
```

```
def scoretable(scores):
    #draw table headers with text
    shapes(15,110,753,60,3,1)
    screen.blit(name_font.render("#",1,white),(30,113))
    screen.blit(name_font.render("Name",1,white),(160,113))
    screen.blit(name_font.render("Score",1,white),(650,113))
    #draw the rest of the table
    for i in range(1,6):
        shapes(15,110+i*80,120,60,3,1)
        shapes(155,110+i*80,475,60,3,1)
        shapes(650,110+i*80,120,60,3,1)
        colour = red
    #blit the high scores, alternating in red and green
    for j in range(len(scores)):
        if j%2==1:
            colour = red
        else:
            colour = green
        screen.blit(name_font.render(str(j+1),1,colour),(30,115+(j+1)*80))
        screen.blit(name_font.render(scores[j][0],1,colour),(170,115+(j+1)*80))
        screen.blit(name_font.render(str(scores[j][1]),1,colour),(665,115+(j+1)*80))
```

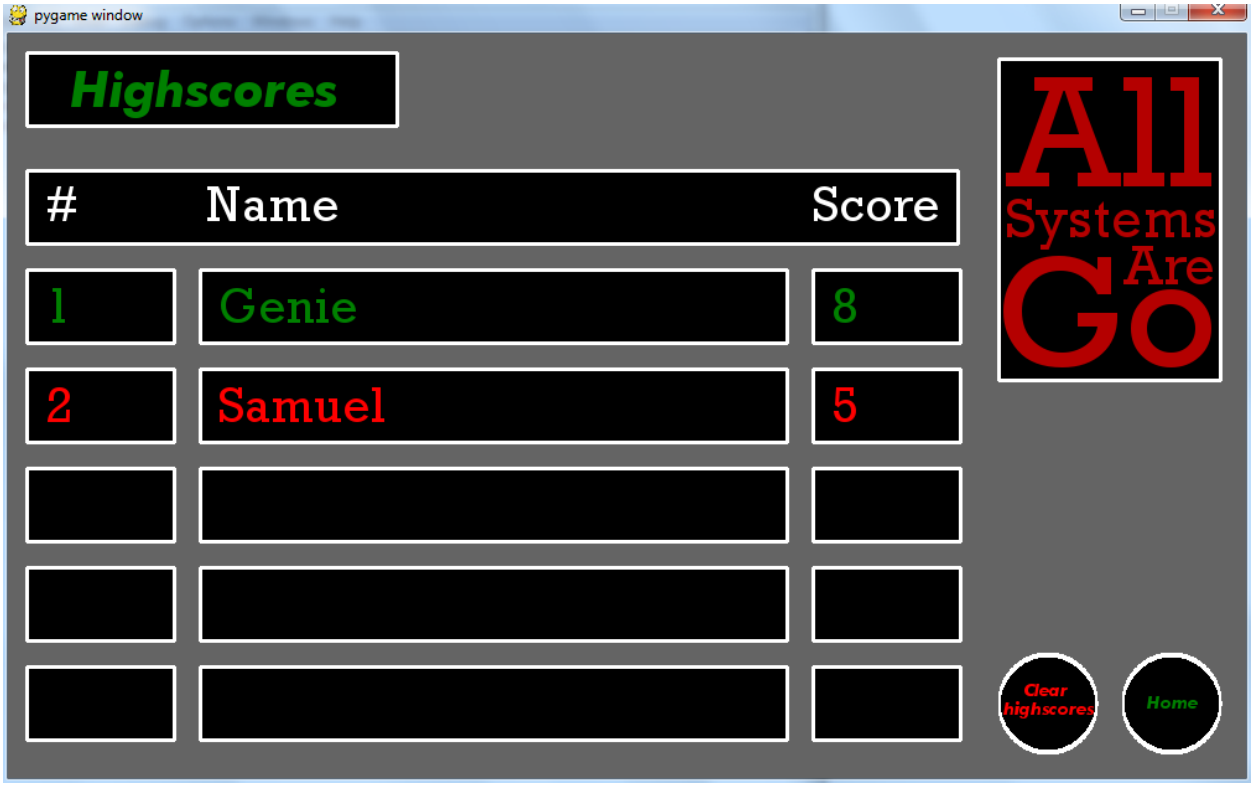
When I ran the program, this was displayed;



The contents of HighscoresFile was empty, so to check if the 'scorestable' module worked, I added some highscores directly to the file. I entered it in the correct order (by score) as I add another module that will sort the scores later.



I ran the program again:



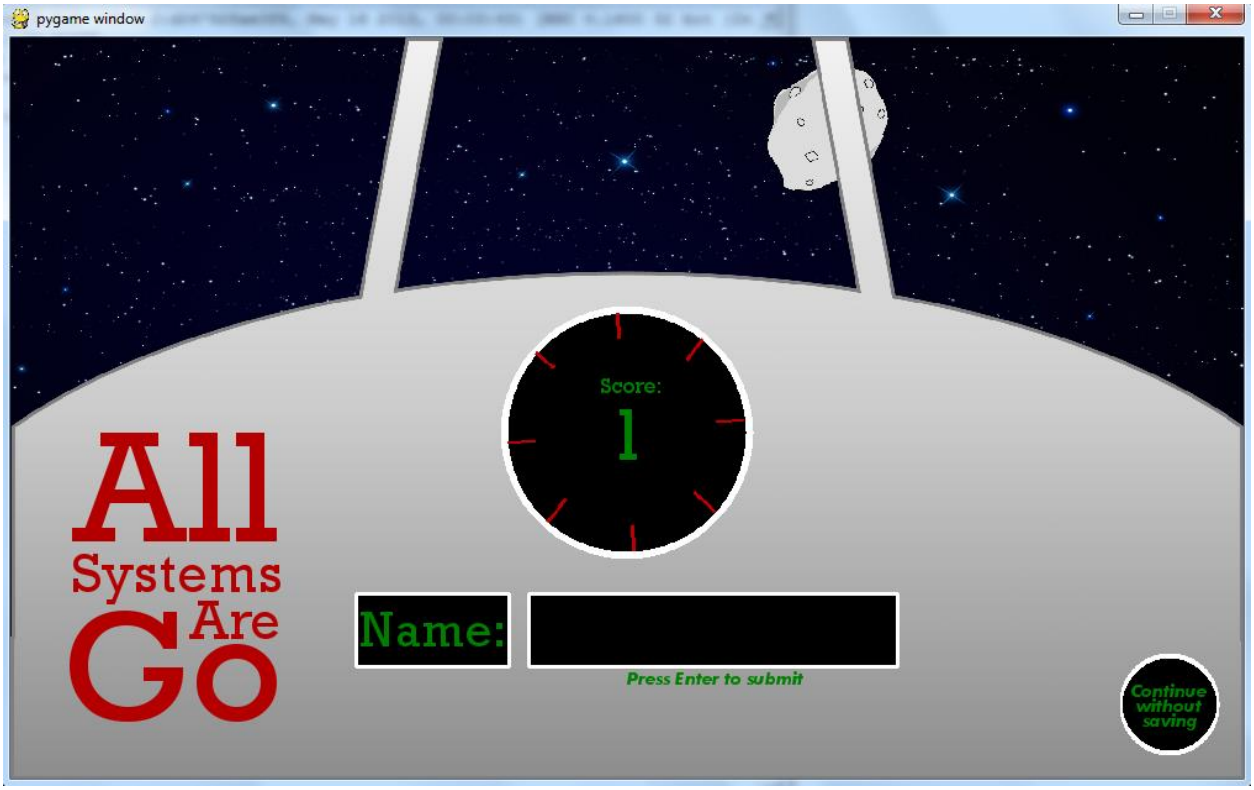
The reading of the file works. When I clicked ‘Clear highscores’, the table was blank again, as was the file.

THE ‘NAMEINPUT’ PAGE

In the “nameinput” section of the main program loop, I added this:

```
    elif page == "nameinput":
        screen.blit(background_img,(0,0))
        #drawing the floating asteroid
        rotation -= 2
        center_x = 400
        center_y = 1000
        radius = 1000
        speed = 0.005
        x = radius * sin(angle) + center_x
        y = radius * cos(angle) + center_y
        angle -= speed
        #call the rotate module for the asteroid
        rotate(spinning_asteroid,spinning_asteroid.get_rect(),rotation,x,y)
        #draw the title and the rest of the background
        screen.blit(name_img,(0,0))
        title(50,285,page)
        #draw the circle showing the score, with the text and rotating red dial
        shapes(400,220,200,200,10,2)
        rotate(dial_img,dial_img.get_rect(),rotation,402,222)
        screen.blit(density_font.render("Score:",1,green),(478,270))
        score = score_font.render(str(level+1),1,green)
        screen.blit(score,[(screen.get_rect().centerx)-(score.get_rect().centerx), 285])
        #draw the home button with the text 'Continue without saving' instead of 'Home'
        home_button = shapes(900,500,80,80,3,2)
        screen.blit(button_font_14.render("Continue",1,green),(908,520))
        screen.blit(button_font_14.render("without",1,green),(913,532))
        screen.blit(button_font_14.render("saving",1,green),(918,544))
        #draw the other boxes and text
        shapes(280,450,125,60,3,1)
        screen.blit(name_font.render("Name:",1,green),(283,455))
        shapes(420,450,300,60,3,1)
        screen.blit(button_font_14.render("Press Enter to submit",1,green),(500,510))
        #blit the name being entered on the keyboard from the center
        display_name = name_font.render(name,1,red)
        screen.blit(display_name,[(screen.get_rect().centerx)-\
                                  (display_name.get_rect().centerx)+ 70, 455])
```

When I ran the program, started a new game, then clicked ‘Save and exit’, this is what appeared:



Everything is where it should be and the red dial marks (and floating asteroid) do rotate. However, nothing that I typed on the keyboard would appear on the screen. This is because in the event loop, I only have commands for mouse clicks and game exiting. So I added this in the events loop:

```
if evnt.type == KEYDOWN: #if the user uses the keyboard
    if page == "nameinput": #the only page where keyboard input is allowed
        if (evnt.key == K_BACKSPACE): #remove the last character if 'backspace' is hit
            if (len(name)>0):
                name= name[:-1]
        elif (evnt.key == K_RETURN): #if 'return' is hit:
            if endgame == True:     #go to the high scores or home page depending
                page = "highscores" #on whether the game was finished or not
                highscores(name, level+1)
            else:
                page = "home"
                save(name,level,lives)
        else: #only allow letters, numbers and spaces in the name
            if (evnt.key >= 48 and evnt.key <= 57) or (evnt.key >= 97 and evnt.key <= 122)\
               or (evnt.key == 32):
                if (len(name)<10): #the name cannot be longer than 10 characters
                    if (key.get_pressed()[K_LSHIFT]) or (key.get_pressed()[K_RSHIFT]):
                    #uppercase letters
                      evnt.key -= 32
                    name += chr(evnt.key) #add character to the end of 'name'
```

Then I ran the program again. This time, the name did appear as I typed it in.

However, pressing Enter (or Return) froze the screen. This is because I had called a module that I haven't made yet:

```
elif (evnt.key == K_RETURN): #if 're
    if endgame == True:      #go to t
        page = "highscores" #on whet
        highscores(name, level+1)
    else:
        page = "home"
        save(name,level,lives)
```

So, I made these modules.

The 'save' module:

```
def save(name,level,lives):
    #append an unfinished game to SavedGamesFile (save game)
    f = open("files/SavedGamesFile.txt", "a+")
    f.write("('" + name + "'," + str(level) + "," + str(lives) +")\n")
    f.close()
```

The 'highscores' module:

```
def highscores(name, score):
    #append a finished game to HighscoresFile (save game)
    f = open("files/HighscoresFile.txt", "a+")
    #add the game to the file
    f.write("('" + name + "'," + str(score) + ")\n")
    f.close()
    #close the file. Then re-open in 'read' mode
    with open('files/HighscoresFile.txt', 'r', encoding='utf-8') as f:
        f.seek(0)
        #copy the contents of HighscoresFile to 'lst'
        lst = [literal_eval(line.strip()) for line in f]
    f.close()
    #reorder 'lst' in order of highest score first
    scoreslst = sorted(lst, key=itemgetter(1), reverse=True)
    #have a maximum of 5 high scores stored
    while len(scoreslst) > 5:
        del scoreslst[-1]
    #re-open file in 'write' mode (blank file)
    f = open("files/HighscoresFile.txt", "w")
    for i in scoreslst:
        #write ordered lst of high scores to file
        f.write("('" + i[0] + "'," + str(i[1]) + ")\n")
    f.close
```

I decided that a maximum of only 5 scores can be stored because the high scores table has 5 positions, and the fewer scores there are, the quicker they are to sort and the less disk space HighscoresFile takes up. (Note: the LOWEST score is deleted)

#test save and highscores modules – WHITE BOX TESTING

TESTING

Df

REFERENCES

http://programarcadegames.com/python_examples/show_file.php?file=sprite_circle_movement.py

http://thepythongamebook.com/en:pygame:step020

http://www.nerdparadise.com/tech/python/pygame/basics/part4/

SECTION 4