Gravity

Group 10

Daniel Walz

Jesper Ekhall

Lukas Kalinski

Fredrik Nordh

Alexander Nyberg

1. Preface

1.1. Version history

2008-01-04 Version 1.0

1.2. Expected readership

The readers of this document are those that are affected by the outcome of the project (they are all stakeholders, but not all stakeholders are readers of this document). The readers falls into two categories which are non technical people and technical people.

Non technical people are people both at the customer or at the developing company. These people are working with things like marketing, business development and sales.

The technical people are the developers at the developing company who needs to understand the detail of what they are about to develop. This document is also intended for technical people at the customer, which needs to understand the details so that they can know whether they got what they ordered or not.

The non technical people will read section 1-4 and 7-9. The technical people will also read section 5 and 6.

Contents

2. Introduction

2.1. Target demographic

Our target demographic would typically be a male in about our age bracket, which is between 12 and 30 years old. This group might for example include a high school student sitting at a computer during breaks or an office worker having job gaps now and then. Targeting primarily casual players who are seeking some entertainment for a limited period of time, the game should be simple in playing and easy to install and start up. Another potential user would be a nostalgic player starting up the game at home either to just kill some time or to play a fun game with a friend. Our product will provide entertainment and as such the main problem it will solve, or at least alleviate, is boredom, regardless of whether it's to brighten up a dreary day at the office or as a complement to a social occasion.

The idea is to provide an easy to play action game, having the outer space as its environment. In that environment there will be planets with gravities to affect other objects such as asteroids and player controlled space ships. Each ship will be able to fly around in that environment and to fire two kind of weapons, targeting an opponent ship and/or asteroids. Making the game more varying, some of the weapon projectiles will be affected by planetary gravities, resulting in a reduced aiming capability and therefore a greater challenge.

2.2. The game concept

In general, Gravity is a game consisting of planets with their corresponding gravities and space ships with the ability to fire at least one weapon. Both space ships and some weapon projectiles will be affected by gravity. The key idea is to combine classic game shooting with gravity constraints, forcing the player to think about more factors than just where to shoot. Keeping the game simple, the view will be presented in 2D and seen from above.

The main usage may be divided into three stages, namely: *game setup*, *game play* and *game exit*.

Game setup will be done using a simple menu, allowing the player(s) to choose which map she would like to play, what game mode (single player or two players) she is interested in and the nickname she would like to use. Choosing multi player mode would also involve choosing what nickname to use for the second player.

Finally, a “getting started” option will also be available, explaining what the game is about and how to control it.

Game play is simply about the game being played, according to the specification mentioned above. The game play may be paused/unpaused by using the pause toggle key. Leaving this stage is done either by pressing the exit key and confirming the exit request or by meeting the game over requirements.

Game exit stage will present statistics about the most recently played session, after which the player will be forwarded to the main game menu.

2.2.1. Use case #1: The two brothers, Peter and Kevin (12 and 13 years old respectively), have a dispute about who is going to use the only computer available. While trying to resolve the situation their father, who recently have heard about a game called Gravity, decides to install it on the computer, remembering that it allows two users to play simultaneously on the same computer. When installed, he quickly sets up a game session by choosing a map and selecting split-screen mode. After a short negotiation Peter and Kevin agree to give it a try. With Peter using alphanumeric steering (i.e. the letters on the keyboard) and Kevin the arrows the game begins. The classic "shoot your opponent" approach, combined with gravities affecting both ships and shots fired, contributes to their mutual satisfaction. While Kevin is chasing his brother between planets, trying to hit him with with the gravity-affected missiles, a new dispute occurs, this time because of their father exiting the game, telling them to go and eat their dinner.

2.2.2. Use case #2: The office employee Eric gets bored by waiting for his boss to give him the green light for a solution he proposed. Therefore he feels free to start playing this game called Gravity, which he recently installed. Eric likes it for its simplicity and for not giving anyone the impression that those who play it are addicted gamers. After starting the game, Eric selects single-player mode, chooses a map, takes a quick look at which the game controls are and what they do, and finally starts the game. Eric really gets into the game and tries to shoot down as many asteroids as possible to get a good score. During the game session Eric's boss comes by, asking him some details about his proposal. Eric hits the pause button and explains what's needed to explain, and when the boss leaves, Eric hits the same button again and continues his game session. A couple of minutes later Eric fails to avoid a planet, as a result of an asteroid coming towards his ship, and crashes. Eric feels that that was enough playing for this time, and exits the game.

2.3. System context and environment

Since the game will be developed for PC computers with Windows XP it will be available in any environment where such systems are in place, for example as mentioned in the use cases at work or at the home computer. For the required machine performance see the appendices. In the case of multimedia computers and laptops together with the broad availability of cheap projectors and capable TV-sets one could also imagine the game being used in a living room setting. Our system will be used on breaks, whenever a user has some spare time over, or in a more social context, for example in a one to one battle with a friend.

2.4. System scope

The game will be played in a 2D environment, depicting the space with a few surrounding planets that attract every other non-planet object according to Newton's physical laws. Each player will be able to control a space ship that can steer left, right and accelerate forward. The ship will be affected by the gravity of the surrounding planets and its steering

will be controlled by the keyboard. There may be more than one player, in which case the players share the keyboard for controlling their respective ships. The ships will be able of firing weapons, which in turn will be able to destroy asteroids or the opponent's ship. If any ship collides with an asteroid or a planet the ship will be destroyed. If an asteroid collides with a planet the asteroid will be completely destroyed. Planets will not be destructible. If any object has reached the frame boundary that the game is played within, it will be teleported to the other side of the axis.

Topic	In	Out
Multi player (split screen)	X	
Multi player (networked)		X
Single player	X	
Mouse control		X
Planets	X	
Moving asteroids	X	
Weapons	X	
Gravity towards planets	X	
2D view	X	
3D view		X
High score system	X	
Moving planets		X
Object collision	X	
Sound effects	X	
Artificial Intelligence(AI)		X

2.5. Design and building factors

Since we are mainly targeting casual players the game should be easy to use and easy to install. This is something that needs great care in the design process because it's very hard to quantify whether a game is easy enough to be played by such a person.

The game must be user friendly, and to achieve this we must take into account things such as having a simple menu available when the game starts up. Among other things, the menu should provide a summary of the game and what the control keys are.

Because the player can get interrupted during game play, a pause function will be available, so that the player can pause the game and continue later.

We are aiming for the system requirements to be pretty low, which we believe shouldn't be very hard to achieve as the game concept is simple comparing to the commercial games available today (2007). However we still need to take performance into account, thinking about how we design the game and at least try to do things in an optimal way.

It can be fun developing a game and it's easy to be carried away by inspiration and try to implement a lot of features instead of thinking about whether the game actually works. Thus we need to focus our attention to building a solid foundation before implementing any extra features.

It is important to make the game entertaining to play, which is another factor that is quite hard to prove achievement on, since there are no strict definitions on what constitutes a fun game. Thus we need to test the game continuously both by ourselves and by prompting others to do so, using the test outcomes to tweak the game so it can reach it's full potential.

2.6. Technologies and risks

2.6.1. Open Graphics Library (OpenGL) – OpenGL is an industrial standard 2D and 3D graphics development platform. We choose it because we have some background knowledge of it and that it’s well tested and available on many platforms. Risks include debugging in graphical systems is difficult, errors are not easy to find. Also not everyone in the development team is experienced in OpenGL programming.

2.6.2. C++ – We choose it because of object orientation, speed, library availability and because it’s easy to use OpenGL from it. C++ is also good to use in environments where garbage collectors might be unwanted because it creates delays. Risks are that it’s a large and complex language. It’s also difficult to debug partly due to pointers and no memory checking. There is also the need to manage memory manually because of no garbage collector.

2.6.3. Windows XP – We choose windows because it’s available virtually everywhere and the team has experience with it. Many useful libraries are available such as OpenGL. Also there is support for I/O with gaming in mind with the DirectX interfaces. Another thing is that it’s easy to use, especially compared to UNIX systems such as Linux. Risks are that it’s a complex system and it requires skill to be stable. The complexity risk is mostly about our users and it’s more of a challenge because we’re targeting casual players rather than experienced gamers.

2.6.4. Simple DirectMedia Layer (SDL) – SDL is a platform independent multimedia library. We are going to use it for taking input from the keyboard and to start up the OpenGL environment. The primary risk is that all members in the group don't have experience with it. Another risk is that it might be to complex or that it won't be complete enough.

2.6.5. Fmod music and sound effects system – Fmod is a portable API for playing music and sound effects. We will use this to play game sound effects during play. Although it's both easy to use and portable, a risk is that we are a bit unsure of the licensing agreements and how they apply to our project.

3. Glossary

2D environment: This is the mode that the game operates in. It says that this game operates in a plane in two dimensions.

C++: C++ is an object oriented programming language. It's a compiled language which has many features but it lacks garbage collection.

Controls: How the game is controlled. For example for keyboard input this are the settings that say how the ship is controlled.

FMOD music and sound effects system: A portable API for playing music and sound effects.

High score list: A list of the best scores achieved.

Installation: A necessary procedure needed before the player can play the game the first time. This procedure puts the software in a state so it can be started easily.

LAN (Local Area Network): A local area network is a local computer network. One of the properties of a LAN is low latencies.

Map: The players world. It's the world that the player interacts with. The map contains information about objects (such as planets), their properties and location in the 2D environment.

OpenGL (Open Graphics Library): This is a library used for interacting with graphics hardware in a portable manner.

SDL (Simple DirectMedia Layer): A platform independent multimedia library which can initialize OpenGL and

Windows XP: A de facto standard operating system made by Microsoft. Features a graphical frontend.

World boundary: Where the playing field ends.

World view: The piece of the map that the player sees while playing.

4. User requirements definition

4.1. Functional requirements

Game configuration

4.1.1. Quick Start Help

A user who doesn't know what the game is about and/or what the controls are, shall be able to get a short summary on these points before starting a game session. The summary shall contain the goals of the game play, together with the currently set controls.

> *Rationale*: Providing a quick start help will enable the actual game play to make a quicker impression on the user, as she doesn't have to struggle and "learn by mistake". The quick start help may also be provided to the user while playing, in case she forgot something.

Test: If it's possible to get the goals of the game play and the controls shown before starting the game, this requirement is met.

4.1.2. Map Choice

Before starting a game session, the user shall be able to choose the map that that session should be played on. If no active map choice is made, the game system shall choose one of the available maps. Each map shall be a definition of the world/environment the player finds herself in while playing a game session.

> *Rationale*: The ability to choose between several maps makes the game more varied as a whole, reducing the risk that the player becomes tired of it too soon. In the case that the player doesn't want to choose a map, the game system may choose one either randomly, by following some constraints or simply by using a default map.

Test: If a default map is chosen by the system when no map is chosen and the user also can chose a map this requirement is met.

4.1.3. Controls Configuration

Before starting a game session, the user(s) shall be able (but not required) to configure the game controls, i.e., what keyboard keys to use for what action in the game. The game shall provide default controls, allowing the user(s) to change them at any time.

> *Rationale*: Even though the default controls should be chosen to satisfy most users' needs, there still will be those that find them inconvenient in some way. Providing the ability to choose controls will satisfy these users.

Test: If it's possible to configure the game controls, this requirement is met.

4.1.4. Single Player or Two Players Choice

Before starting a game session, the user shall be requested to choose whether she wants to play in single player mode or to split up the screen, enabling two users to play against each other. In the case of a "two players" choice, it shall be possible to set controls for and names of both players.

> *Rationale*: Providing both single player and two players modes widens the contexts within which the game can be played.

Test: If it's possible to choose single or multi player mode this requirement is met.

4.1.5. Two Players Game Rule Choice

Before starting a game session in two players mode, the users shall be able to set a points limit, defining when the game is going to end. The game shall end when the set point limit is reached by any of the two players.

> *Rationale*: Setting a "game end" rule like this is a convenient way for the users to restrict their playing time, and on a larger scale it'll prevent that they eventually remove the game as a result of uncontrollable (addictive) playing.

Test: If it's possible in two player mode to choose points limit and game ends after set number of points this requirement is met.

4.1.6. Single Player High Score List

Exiting the game, either as a result of losing all ships (game over) or by exiting the game deliberately, shall let the player know about the current high score list and, if she had qualified for a placement on it, request her name. Only exit options provided by the game shall follow this requirement.

> *Rationale*: Having a high score list will give a more "permanent" goal to a game session, motivating the player to strive for a concrete result.

Test: If the player has reached a number of points enough for the high score list, the player should be prompted for her name. Also the high score list should be shown after a game ends. If both this occurs, this requirement is met.

4.1.7. Exiting The Game

It shall be possible to exit the game at any stage. While not playing the game, the user shall be able to quit to the operating system. The user shall be prompted if she is sure she wants to quit, when any exit function is chosen. While playing, the user shall be able to choose whether to exit to setup or exit to the operating system.

> *Rationale*: Being able to exit a game is a fundamental feature, which simply enables the user to decide when she wants to exit and to do so properly.

Test: If it's possible to exit to setup and to operating system while playing the game and also to exit to operating system while in setup this requirement is met.

Game play

4.1.8. World Boundary Wrapping

When a player makes her ship to go beyond one of the world boundaries, it shall be "teleported" to the opposite side of the world. For example, going out on the left side shall result in appearing on the right side.

> *Rationale*: As the alternative was to "lock in" the ships in the world, which would result in a restricted area of movement, the idea of simulating open space (which in fact is what the game world is supposed to represent) will give the user more freedom about which paths she would like to follow.

Test: When the ship go outside the upwards the ship should restart from the bottom and opposite. When the ship goes outside to the left of the screen the ship should restart from the right side of the screen. If both these are met, the requirement is met.

4.1.9. Player's World View

The player shall see her ship from above, at a distance that depends on the ship's movement speed. When the speed is increasing, the viewing distance shall increase too. Conversely, when the speed is decreasing, the viewing distance shall decrease too.

> *Rationale*: Providing a speed dependent viewing distance will allow the player to see possible obstacles in time when moving fast, and to see more detail and therefore gain some game feeling when moving at a lower speed.

Test: If the zoom-level is proportional to the speed of the ship (more zoomed out the faster the ship moves), this requirement is met.

4.1.10. Game Play Information

The player shall be able to see information about her ship's health and fuel amount during the game play. If the game rules say that there should not be any fuel restrictions (i.e., unlimited fuel supply), then information about the fuel amount should be left out.

> *Rationale*: Allowing the player to see information about her ship will enable her to apply strategies depending on what the information is saying.

Test: If ship information is visible while playing the game, this requirement is met.

4.1.11. Scoring in Single Player Mode

Scores shall be gained partly by shooting at asteroids and partly by the time the player managed to stay alive, having three ships (and therefore chances) at her disposal.

Rationale: As the main challenge of playing in single player mode is to avoid collisions, fire down as many asteroids as possible and stay alive as long as possible, this is a matching scoring strategy, rewarding the player for taking care of these challenges.

Test: If points are gained slowly while travelling around, and faster when shooting an asteroid this requirement is met.

4.1.12. Scoring in Two Players Mode

A player shall get rewarded when destroying his opponent's ship with any weapon. A player shall be punished when his ship is destroyed by crashing into some obstacle in the world (including the opponent's ship).

Rationale: Getting scores for killing the opponent and losing scores when "suicide" crashing is a reasonable scoring strategy. Conversely, getting scores for killing and losing scores *when killed*, could, in case of equally experienced players, result in a gaming session lasting forever, which would be in conflict with the purpose of the scoring feature (which is to reduce the gaming time to some extent).

Test: If the player get points when destroying an opponent, and gets lower score when being killed, this requirement is met.

4.1.13. Single Player Ship Disposal (Lives)

The player shall start having three ships at her disposal. Gaining a certain amount of points shall give another ship. The ships shall not be used simultaneously, but once one is crashed it shall be replaced with a new one if available, otherwise the game ends and the achieved points shall be displayed together with a high score list.

Rationale: Restricting the number of ships to expend allows the game to end at a certain point, which is preferable due to the aims of this game – to be a time killer or short entertainment, not a long never ending game session.

Test: If the game information displays three ships when the game starts, and if one ship is removed when the players ship is destroyed, and if one ship is added when a preset amount of points are collected this requirement is met.

4.1.14. Ship Speed Restriction

A ship's movement shall be restricted in speed. When the speed reaches a set limit, throttling shall not be able to increase it.

Rationale: Not having a speed limit for the ship would result in uncontrollable speed and therefore less fun game play. A speed limit will allow the player to have throttle on all the time, without risking to lose control.

Test: If a player accelerates and reaches this speed no further thrust will accelerate the ship then this requirement is met.

4.1.15. Ship Fuel Restriction

A ship shall either have a fuel restriction (when in single player mode) or have an infinite fuel supply (when in two players mode). When there is a fuel restriction, it shall also be a restriction on how much fuel the ship can have at once.

> *Rationale*: In single player, the fuel restriction will add some challenge in complement to the shooting at and avoidance of asteroids. However, in two players mode the fuel restriction would be rather inconvenient for the player, and therefore it is sufficient with (and even without) asteroids there, as there are two human players playing against each other, having that battle as their primary goal.

Test: If the fuel amount reaches zero no further thrust will be possible. In two player mode no amount of thrust will decrease the amount of fuel available. If both this are achieved the requirement is met.

4.1.16. Ship Damage

A ship shall be completely damaged when colliding with other ships, a planet or an asteroid. A ship's damage resulting from a weapon projectile hit shall be defined by the destructive power of that projectile. Complete damage results in ship destruction.

> *Rationale*: It is reasonable to consider each non-projectile collision as completely destructive for the ship, as the objects that are being collided with are much more solid than a ship. Conversely, a projectile does not always have to completely destroy a ship when hitting it, at least if not being very powerful. Allowing ships to be damaged forces the player to avoid both obstacles and, when in two players mode, the opponent's weapon projectiles.

Test: The ship shall be destroyed after colliding with a ship, planet or asteroid and the ship shall also resist one shot without being completely damaged, if so this requirement is met.

4.1.17. Operating a Ship

A ship shall be controllable by throttling (i.e., gaining speed in the direction of the ship) and steering right and left respectively. Once a ship's movement and speed is achieved it shall remain constant until its destruction, unless affected by a gravity or its throttling.

> *Rationale*: These controls are sufficient to control a ship. Adding a brake control could be an option, although as we're in space, no such thing should exist. Simulating brakes can be done by turning the ship half a turn and throttling. The ability to control a ship is a basic and expected functionality of this game and therefore it should be provided.

Test: If the ship is maneuverable this requirement is met.

4.1.18. Ship Laser Gun

A ship shall be able to fire laser projectiles. The laser projectiles shall not be affected by planetary gravities. The damage caused by a laser shall be partial. Lasers shall travel fast (in relation to missiles).

> *Rationale*: Since lasers are not affected by gravities, it will be much easier (than with missiles) to hit a target when using them, and therefore their damage power should be reduced. The exact level of damage a laser projectile can cause should be defined at a later stage. The laser gun will enable the player to hit his targets more easily.

Test: If a ship can fire laser projectiles this requirement is met.

4.1.19. Ship Missile Launcher

A ship shall be able to fire missile projectiles. The missile projectiles shall be affected by planetary gravities. The damage caused by a missile shall be complete. Missiles shall be slower than a laser projectile, but faster than a ship.

> *Rationale*: Since missiles are affected by gravities, it will be hard to hit a target when using them, and therefore their damage power should be complete. The missile launcher will enable the player to cause complete destruction when succeeding in hitting a destructible target.

Test: If a ship can fire missile projectiles this requirement is met.

4.1.20. Operating a Ship's Weapons

The player shall be able to choose which weapon to use before firing it off. The projectile resulting from firing a ship's weapon shall be set off from the ship's current position and in the current direction of the ship (i.e., not ship movement, but where the ship will strive to go when/if throttling).

> *Rationale*: Firing a weapon in the direction of the ship is the most natural thing to do, as this is the expected behavior in this kind of games. The player will therefore not have to control both the ship and the ship's guns, but will be able to control both at the same time, making the game play more simple.

Test: If a player can choose a weapon before firing it, this requirement is met.

Passive objects

4.1.21. Planets

A planet shall have a gravity which shall affect ships, missiles and asteroids exclusively. A planet shall not move in any way. An object being affected by a planet's gravity shall be pulled towards that planet with a certain strength. Asteroids hitting a planet shall, if that is the rule of the game mode or the map, increase the planet's gravitation.

Rationale: Having planets with corresponding gravities makes these more than just simple obstacles to avoid, resulting in a more challenging game play.

Test: If planet attracts ships and asteroids this requirement is met.

4.1.22. Asteroids

In single player mode, asteroids shall be sent in to the world at an adequate frequency, making the game play challenging enough. In two players mode, asteroids may be sent in at a deliberate frequency. Asteroids shall be destructible, both partially and completely. Partial destruction means that an asteroid is split into two smaller asteroids, while complete destruction means that the whole asteroid is destroyed.

Rationale: Having asteroids is most crucial in single player mode, as these, together with increasing gravities (as a result of an asteroid hitting a planet) and picking up fuel items, will be the only challenge for the player.

Test: If asteroids appear within 1 minute after the game starts this requirement is met.

4.1.23. Items

In single player mode, items containing fuel shall occur randomly in both place and time in the world, and frequently enough to guarantee that the player has a fair chance to pick them up before running out of fuel. In two players mode, items shall not contain fuel, but instead they shall contain weaponry and ship health upgrades. Items shall appear in free space in the world, allowing a player to pick them up.

Rationale: In single player mode, items will force the player to not stay in "safe places" (i.e., where no asteroids hit) but to constantly search for more fuel and therefore actively participate in the game play. In two players mode, items may add some value to the game, by complementing it with unpredictable ship and weaponry upgrades. The usage of an item may be constrained by either time or amount, especially when it makes the owning player more or less invincible.

Test: If an item appears within 10 minutes from game start this requirement is met.

Misc

4.1.24. Sound Effects

Sound effects shall be played for each of the following events: collisions, fired weapons, ship throttle and item pickups.

Rationale: Adding sound effects to this game will make it much more alive.

Test: When the event occurs the sound shall be heard then this requirement is met.

4.2. Use cases

4.2.1. Play a single player game

Primary actor: The player

Stakeholders and interests: Player - Computer left in a sane state and the player has entered the high score list.

Preconditions: Game installed and game instance started.

Success guarantee (postconditions): Successfully started and played a single player game and entered a high score.

Minimal guarantee: The player has had the opportunity to maneuver the ship.

1. The player starts a single player game.

2. The player maneuvers the ship.

3. The player fires a shot aiming towards an asteroid.

4. The system responds with an asteroid exploding.

5. The player loses all his ships and enters high score.

6. Use case ends.

Extensions:

*a. At all times, if a ships energy level reaches zero, player loses a ship.

*b. At all times, if the player has lost all his ships the game ends and the player enters high score, the use case ends.

2a. Player collides with a planet.
   1. System withdraws energy from ships health.

2b. Player collides with an asteroid.
   1. Systems withdraws energy from ships health.

4a. The shot hits a planet.
   1. The shot is absorbed by the planet.

4b. The shot misses the asteroid and wraps around space and hits the player ship.
   1. Systems withdraws energy from ships health.

Variations:

1. Shot is

   a. a laser shot.

   b. a missile shot.

4.2.2. Play a multi player game

Primary actor: Player one.

Stakeholders and interests:

Player one - Wants to defeat player two.
Player two - Wants to defeat player one.

Preconditions: Game installed and game instance started.

Success guarantee (postconditions): Successfully started and played a multi player game and destroyed player two.

Minimal guarantee: The players has had the opportunity to maneuver their ships.

1. Player one starts a multi player game.

2. The players maneuver their ships.

3. Player one fires a shot aiming towards player two.

4. The system responds by withdrawing energy from player two's ship.

5. Player two is destroyed and the use case ends.

Extensions:

*a. At all times, if a ships energy level reaches zero, the player loses a ship.

*b. At all times, if the player has lost all his ships the game ends and the use case ends.

2a. Player collides with a planet.
   1. System withdraws energy from ships health.

2b. Player collides with the opponent.
   1. Systems withdraws energy from both ships health.

4a. The shot hits a planet.
   1. The shot is absorbed by the planet.

4b. The shot misses player two and wraps around space and hits player one's ship.
   1. Systems withdraws energy from the ships health.

Variations:

1. Shot is

   a. a laser shot.

   b. a missile shot.

4.2.3. Enter high score

Primary actor: The player

Stakeholders and interests:
Player: Player wants to enter into high score list, if he has enough points.
Computer owner: No files outside of game directory altered.

Preconditions: Game installed and game instance ended.

Success guarantee (postconditions): High score entered.

Minimal guarantee: High score validated and high score list shown.

1. The system validates the player's score.

2. The system prompts the player to enter his name

3. Player enters his name.

4. The system validates name.

5. The system saves high score list.

6. The system views the high score list.

Extensions:

1a. The system detects that the player's score is not enough for high score list.
   1. The system views the high score list.

4a. The system detects non-legal characters.
   1. The system prompts the player to enter his name again.

5a.The system doesn't have enough system resources to save the high score list.
   1. The system views the high score list.

5. System architecture

5.1. High level overview

5.1.1. Module: Controller

This module will contain everything that is related to user input and its consequences for the system flow.

5.1.1.1. Class: Input Manager

With future support for networking in mind, this is a crucial class within this module. It is able read input from anywhere as it relies on an interface (InputListener) instead of hard-coded one-purpose input listening.

5.1.1.2. Class Group: Game Controllers

The game controller classes are responsible for issuing commands on different areas within the game, such as for example on a player's ship or the main menu.

5.1.2. Module: Game

This module will contain game logic, which includes the different states (menu, game play, etc.) and their logic, for example, the game play state will use the Physics Engine.

5.1.2.1. Class: Gravity

The starting point of the system, forwards main loop to different game states.

5.1.2.2. Package: State

Contains representations of the states that the system may fall into. Each state is responsible for issuing a render of its own view. In addition, the game play state will also be responsible for issuing audio playback.

5.1.2.3. Package: Engine

Contains everything that is related to calculations within the game world and the menus. This for example includes the Physics Engine.

5.1.3. Module: Audio

This module will contain classes that are responsible for playing sounds.

5.1.3.1. Class: Distributor

The Distributor is the heart of the audio module. It is responsible for "delivering" the sound to wherever it is to be delivered, relying on the Supplier interface. This property makes sound playback easy to integrate with a possible future networking environment.

5.1.3.2. Interface: Supplier

Defines the interface of an audio supplier, which basically is a class that is responsible for making one or more sounds heard somewhere.

5.1.4. Module: View

This module will contain classes that are responsible for rendering and displaying graphics.

5.1.4.1. Class: Distributor

The Distributor is the heart of the view module. It is responsible for "delivering" a graphical view to wherever it is to be delivered, relying on the Supplier interface. This property makes view rendering easy to integrate with a possible future networking environment, for example by having a class for sending view data implement the Supplier Interface.

5.1.5. Module: Data Store

This module will maintain shared data.

5.1.5.1. Package: Config

Will contain classes responsible for reading and writing configuration files.

5.1.5.2. Package: Menu

Will contain classes responsible for holding data about the game menus.

5.1.5.3. Package: Player

Will contain classes responsible for holding data about each player.

5.1.5.4. Package: GameWorld

Will contain classes responsible for holding data about the game world environment (such as planets, asteroids, etc).

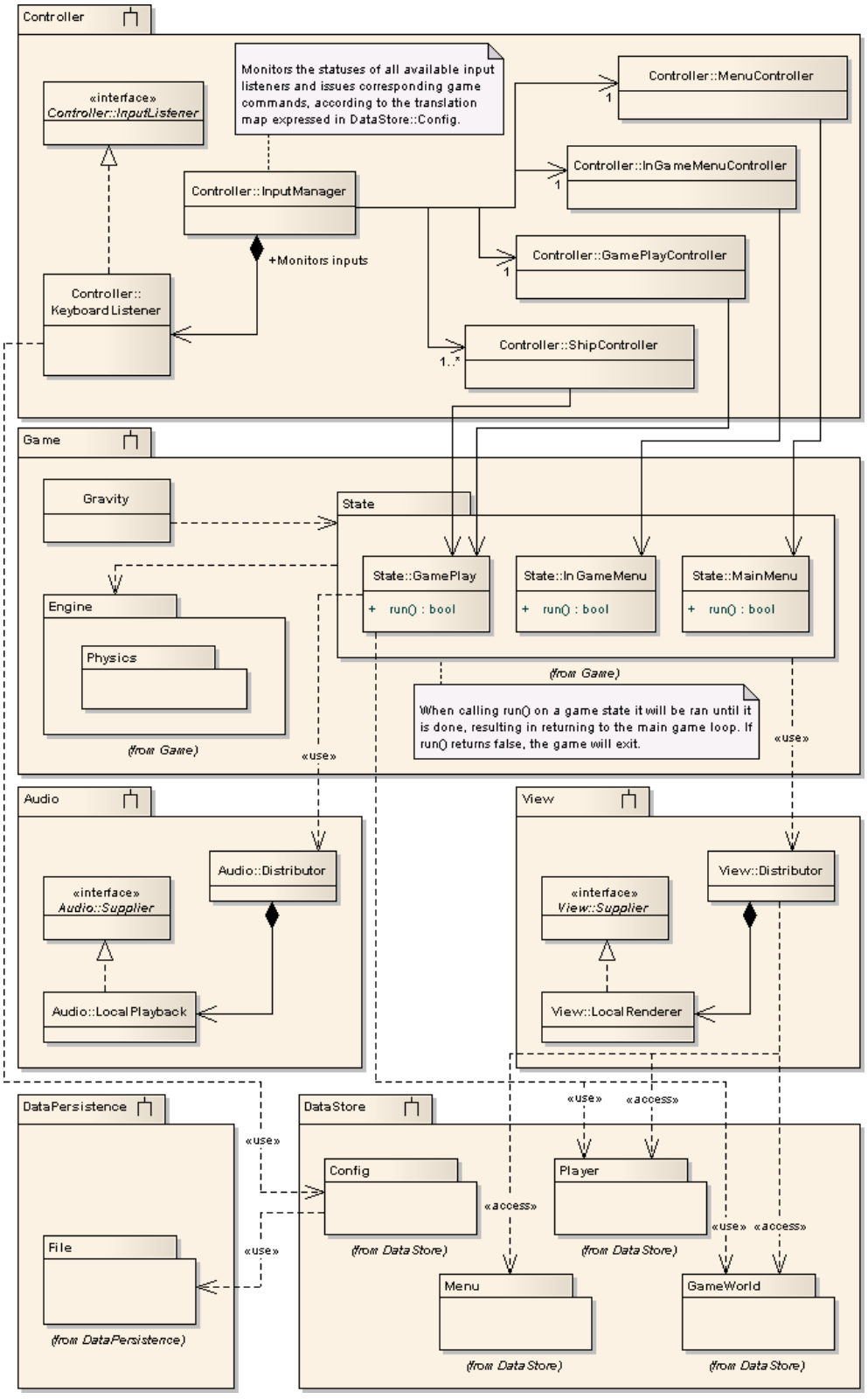
5.1.6. Module: Data Persistence

This module will contain classes that are responsible for loading and saving persistent data.

5.1.6.1. Class: File

Class responsible for file management (read, write, etc).

5.2. Overview chart



6. System requirements specification

6.1. Non functional requirements

6.1.1. C++

Central parts of the system depend on the use of object oriented C++ and it shall be used as the only programming language for this software project.

6.1.2. OpenGL

The user interface shall be in 2 dimensional space and implemented using the OpenGL graphics library.

6.1.3. Sound

The sound system shall use the FMOD API.

6.1.4. Windows XP

The system must be executable on the Windows XP platform.

6.1.5. 24 picture updates per second

The system must be able to update the game window at least 24 times per second on a moderately powered system. This goal will be considered fulfilled if it runs at specified performance on a computer with AMD Turion 64 X2 1.6GHz, 2GB of RAM, and ATI mobility radeon X1300 graphics card or a computer with P4 mobile 2.0GHz, ATI mobility radeon X700.

6.1.6. Installation

Installation shall be very simple. The system shall not require to be shipped as an executable installation file but be contained in a zipped file then unzipped to a directory and runable from there without any further requirement.

6.1.7. Write access to file system

The software package requires read and write access to the file system, in particular the directory where the game resides.

6.1.8. Easy too use

It must take an average person within the demographic no more than 15 minutes to be able to use the game.

6.1.9. Maximum hard-drive usage

The game shall not take more than 100MB of hard-drive space.

6.1.10. Development process

In the development process we shall use the trac software project management tool, the SVN source code version control system and the default compiler shall be GCC.

6.2. Functional requirements

Game configuration

6.2.1. Quick Start Help

Function: Provide basic instructions to the user.

Input: None.

Output: None.

Initiated by: User event.

> Action: The user shall be presented with information about the goal of the game and how the game is played. Information about game controls shall be queried from the user settings and displayed. Design wise this information must be presented in a manner congruent with the overall design of the game (i.e. it must not open a third part text editor.) If a game is in progress when the user initiated this function the game shall be paused and upon return the game shall be unpaused.

Pre-conditions: None.

Post-conditions: None.

Side effects: None.

6.2.2. Map Choice

Function: Choose map that the next game session will be played on.

Input: Various game maps that are available.

Source: From consistent storage.

Output: A map identifier.

Initiated by: Game event.

Action: The user shall be presented the choice between the maps that are available on the system. If the user does not choose, a default map shall be used. The maps shall be provided with the game itself in the form of external files, these files shall conform to a standard format and shall contain a specification of: the number of planets, their positions, the ships' starting position(s) and how many and how fast the asteroids in the world are.

The map file shall define the following features of the game:

1. Number of planets, for each planet describe position, radius and mass.

2. Maximum number of asteroids on screen at any time.

3. Asteroid arrivals per minute.

4. Average velocity of asteroid.

Pre-conditions: None.

Post-conditions: A map will be selected.

Side effects: None.

6.2.3. Controls Configuration

Function: Specify controls used throughout the game.

Input: None.

Output: Matching between game input and game actions.

Initiated by: User event.

Action: The user shall be able to define what game inputs are used to control certain game actions. These configurations shall be stored on consistent storage in a file. The file shall contain the keyboard configuration for both player 1 and player 2, describing the following actions: throttle, turn left, turn right, fire, change weapon.

Pre-conditions: Write privileges to storage device.

Post-conditions: None.

Side effects: Outputs configuration file to storage.

6.2.4. Single Player or Two Players Choice

Function: Choose mode for the next game session.

Input: None.

Output: Specification of game mode.

Initiated by: Game event.

Action: The user may choose between a single and a two player game session. This choice will carry over and define the next game session that is played.

Pre-conditions: None.

Post-conditions: None.

Side effects: None.

6.2.5. Two Players Game Rule Choice

Function: Set rules for the next game session.

Input: None.

Output: Set of rules.

Initiated by: Game event.

Action: The user shall be provided with a way to specify the information described in the user requirements section. These will be stored and used in the next game session.

Pre-conditions: A two player game was selected (see above.)

Post-conditions: None.

Side effects: None.

6.2.6. Single Player High Score List

Function: Display and update high score.

Input: User game score.

Output: Modified high score list.

Initiated by: Game event.

Action: If the users score is high enough (relative to the existing scores) this function will ask the user for a name and will then enter this information to the high score list. The high score list is a set of two-tuples made up of a name and the corresponding score where the name is a string of letters and the score is an integer, each two-tuple is designated a position in the list such that the tuple in the first position has the highest value in score, the tuple in the second position has the next highest value etc. A new entry to this list is added in the same manner. If an added tuple causes the cardinality of the set to be above a preselected limit the tuple with the lowest score will be bumped off the list.

The high score list shall be saved to consistent storage in a file containing each tuple as described above and displayed to the user such that each tuple and their relative position is reflected in the order of the list.

Pre-conditions: Write privileges to storage device. User has finished a game session.

Post-conditions: None.

Side effects: Outputs file to storage.

6.2.7. Exiting The Game

Function: Exit the game.

Input: None.

Output: None.

Initiated by: User event.

> Action: Exits a game session or the game itself, as specified in the user requirements.

Pre-condtions: None.

Post-conditions: None.

Side effects: None.

Game play

6.2.8. World Boundary Wrapping

Function: Defines movement near edges of the playing field.

Input: None.

Output: None.

Initiated by: Game event.

> Action: The playing field is defined by a 2-dimensional plane, if the player reaches the boundaries of this plane then the player shall emerge at the other extremum of the plane. For example, assume that the plane is bounded by -y,y,-x and x in a Cartesian coordinate system if the player reaches the coordinate -y with any corresponding value for the horizontal position the player will emerge at y with an unchanged horizontal position. The same applies to movement along the vertical axis and a combination of the two.

Pre-conditions: A game session is active. The player has reached a boundary.

Post-conditions: The player has the correct position.

Side-effects: None.

6.2.9. Player's World View

Function: Change players view of the playing field.

Input: Speed of the ship.

Output: Camera position.

Initiated by: Game event.

Action: If the players ship is moving at a speed v, the distance between the camera and the plane (as measured from the camera to the plane along a normal from the camera) shall be a function of v. This function shall be defined between a constant nearest distance, Z_near, and a constant farthest distance, Z_far.

Pre-conditions: A game session is active.

Post-conditions: None.

Side-effects: None.

6.2.10. Game Play Information

Function: Provide real-time information about the current game-session.

Input: None.

Output: Game information.

Initiated by: Game event.

Action: The information defined in the user requirements section shall be visible during a game-session, the information shall be presented in such a way that it is plainly and easily accessed during each stage of the game-session.

Pre-conditions: A game session is active.

Post-conditions: None.

Side-effects: None.

6.2.11. Scoring in Single Player Mode

Function: Calculate score in a current game session.

Input: information about game session.

Output: Current score.

Initiated by: Game event.

Action: The score shall be calculated as defined in the user requirements. The score is accumulated throughout a game session and reset after a game session if over. The actual score shall be represented by an integer (both positive and negative) and start at zero.

Pre-conditions: A game session is active. The game session is in single player mode.

Post-conditions: None.

Side-effects: None.

6.2.12. Scoring in Two Players Mode

Function: Calculate score in a current game session.

Input: information about game session.

Output: Current score.

Initiated by: Game event.

Action: The score shall be calculated as described in the user requirements section. The scoring system shall also tell the system when a certain scoring limit is met, if any was set, resulting in the end of the game session (this shall only apply to two player mode). The actual score shall be represented by an integer (both positive and negative) and start at zero.

Pre-conditions: A game session is active. The game session is in two player mode.

Post-conditions: None.

Side-effects: None.

6.2.13. Single Player Ship Disposal (Lives)

Function: Keep track of players lives

Input: None.

Output: None.

Initiated by: Game event.

Action: The number of lives shall be represented by a non-negative integer, if a ship is destroyed (refer to the Ship Damage requirement,) one life will be lost. If the number of lives reaches zero the game session shall end (refer to the Use Cases.) The maximum number of lives is a non-configurable entity.

Pre-conditions: A game session is active.

Post-conditions: None.

Side-effects: None.

6.2.14. Ship Speed Restriction

Function: Keep track of and restrict ship speed.

Input: Ship velocity.

Output: None.

Initiated by: Game event.

Action: If the ships velocity is denoted, in vector form, (vx,vy) then the magnitude of this vector, or in other words: the speed, can not change in excess of a preset limit. That is, no amount of throttle can increase the magnitude of the velocity vector. Note however that throttling in a manner that decreases the current speed is of course allowed.

Pre-conditions: A game session is active.

Post-conditions: None.

Side-effects: None.

6.2.15. Ship Fuel Restriction

Function: Keep track of fuel consumption.

Input: None.

Output: None.

Initiated by: Game event.

> Action: A real number denoting the amount of fuel available shall be associated to the ship. This amount is increased by picking up items (refer to the Items section of the requirements.) The amount of fuel is decreased if the throttle function is used, this will result in a force on the ship causing an acceleration. The force shall be a function of the amount of fuel consumed and the time in which it is consumed. If the amount of fuel reaches zero the player shall not be able to throttle and the ship shall maintain the current velocity.

Pre-conditions: A game session is active. The current game session is in single player mode.

Post-conditions: None.

Side-effects: None.

6.2.16. Ship Damage

Function: Decrease ships shield.

Input: None.

Output: None.

Initiated by: Game event.

> Action: All ship will have an associated value denoted shield that defines how much damage a ship can receive without being destroyed. This value is decreased as defined in the User Requirements section. If this value reaches zero the player will lose one life (refer to the Single Player Ship Disposal (Lives) Requirement.)

Pre-conditions: A game session is active.

Post-conditions: None.

Side-effects: None.

6.2.17. Operating a ship

Function: Allow player to control a ship object.

Input: None.

Output: None.

Initiated by: User event.

Action: Using the defined controls (refer to the Controls Configuration requirement) the player shall be able to throttle and spin the ship. The throttle shall increase the ships velocity vector in the direction defined by the ships nose (or direction vector,) in the two player mode the acceleration of the ship is defined as a function of the time the throttle is applied (restricted by the Ship Speed Restriction requirement) and in the single player mode the acceleration is as defined in the Ship Fuel Restriction requirement.
The turn function shall spin the ship around its central axis in the plane determined by the playing field, this will change the direction vector. Each application of the defined turn control shall spin the ship a preset number of degrees.

In the case of the player triggering "turn left" and "turn right" controls simultaneously, the system shall only consider the one which was registered by the system in the first place.

Pre-conditions: A game session is active.

Post-conditions: None.

Side-effects: None.

6.2.18. Ship Laser Gun

Function: Fire a laser pulse.

Input: None.

Output: None.

Initiated by: User event.

Action: If the fire weapon button is pushed (refer to the Controls Configuration requirement) and the Laser weapon is selected (refer to the Operating a Ship's Weapons requirement) a laser pulse shall be fired in the direction of the ships direction vector and originating from the ships current position. The speed of the pulse is preset and is relative to the ship that fired the pulse. The damage caused to another ship shall be defined by a preset number and the damage to asteroids is described in the Asteroids requirement.

Pre-conditions: A game session is active.

Post-conditions: None.

Side-effects: None.

6.2.19. Ship Missile Launcher

Function: Fire a missile.

Input: None.

Output: None.

Initiated by: User event.

Action: If the fire weapon button is pushed (refer to the Controls Configuration requirement) and the missile weapon is selected (refer to the Operating a Ship's Weapons requirement) a missile shall be fired in the direction of the ships direction vector and originating from the ships current position. The speed of the projectile is preset and is relative to the ship that fired it. The missile shall completely destroy other ships and the damage to asteroids is described in the Asteroids requirement.

Pre-conditions: A game session is active.

Post-conditions: None.

Side-effects: None.

6.2.20. Operating a Ship's Weapons

Function: Use and change weapons.

Input: None.

Output: None.

Initiated by: User event.

Action: The player shall be able to cycle weapons using the keyboard. The cycling process will proceed as follows: the user presses the cycle key (refer to the Controls Configuration requirement) and the game displays which weapon is currently equipped, if the user presses the fire button a projectile corresponding to the selected weapon shall be fired. The in game weaponry should consist of at least the following; missiles and lasers.

When a player presses the firing button on the controls a projectile will be sent out in a straight direction from the nose of the ship. The projectile will continue forward until it hits an object in the world.

Pre-conditions: A game session is active.

Post-conditions: None.

Side-effects: None.

Passive Objects

6.2.21. Planets

Function: Define planets.

Input: None.

Output: None.

Initiated by: Game event.

Action: A planet is a game world object defined by: a position, a radius and a mass these properties are defined by the map (refer to the Map Choice requirement.) A planet shall attract other game world objects (ships and asteroids) with a force proportional to the mass of the planet and the mass of the object.

The planets shall be stationary world objects. Planets cannot be destroyed by the player. The players ship shall be destroyed if it crashes into the planet (that is if the ship is on or within the boundary of the planet.) Asteroids can also crash into planets and in that case they are completely destroyed. The projectiles from a players weapons are destroyed against planets but have no effect.

Pre-conditions: A game session is active.

Post-conditions: None.

Side-effects: None.

6.2.22. Asteroids

Function: Define asteroids.

Input: None.

Output: None.

Initiated by: Game event.

Action: An asteroid is a game world object defined by: a position, a radius, a mass, a force and a velocity. The force is derived from the gravitational effect from surrounding planets and this force shall affect the velocity. A new asteroid is always introduced from one of the edges of the playing field and the position is then determined by accumulating the velocity. An asteroid is introduced meeting the requirements of the current map (refer to the Map Choice requirement.)

There shall be three kinds of asteroids; full sized, half size and quarter sized. If a full sized asteroid is hit it will shatter into a preset number of half-sized asteroids and if half-sized asteroids are hit they shall shatter into a preset number of quarter-sized ones. If the quarter sized asteroid is hit it will disintegrate completely.

If the asteroid is hit by a laser weapon (refer to the Ship Laser Gun requirement,) that is if the pulse is on or within the boundary of the asteroid, it will shatter into pieces as described above.

If the asteroid is hit by a missile (refer to the Ship Missile Launcher requirement) then the asteroid will disintegrate completely regardless of the size of the asteroid.

Pre-conditions: A game session is active.

Post-conditions: None.

Side-effects: None.

6.2.23. Items

Function: Define item objects.

Input: None.

Output: None.

Initiated by: Game event.

> Action: An item object is defined by the following properties: A position and an item. The item property defines how the item aids the player (for example an item can be additional fuel.)
>
> An item (as defined in the User Requirements section) shall be picked up by colliding with the item object, the item will then be acquired. There shall be two classes of items, automatic and manual. The automatic items (for example fuel) shall be applied directly when the item is picked up (when the player acquires the fuel item the fuel amount shall be added to the players fuel supply without intervention of the player.) Manual items (for example weapons) shall be added to the players inventory and requires the player to actively use the item.
>
> The objects that represent items shall appear at random intervals in open space (i.e. not on a planet.)

Pre-conditions: A game session is active.

Post-conditions: None.

Side-effects: None.

Misc

6.2.24. Sound Effects

Function: Play sound effects.

Input: Sound effect.

Output: None.

Initiated by: Game event.

> Action: For each of the events in defined in the User Requirements section a sound shall be played, this sound will be stored on consistent storage and provided with the game. The sound effect shall be played using the FMOD API.

Pre-conditions: None.

Post-conditions: None.

Side-effects: None.

7. System evolution

7.1. Networked multiplayer mode

A possible extension to the game would be to allow multiple players play against eachother over the internet. We think this would be a great addition of gameplay value, but would be relatively labour intensive. A possibly easier extension would be networked multiplayer mode over a LAN only. With LAN only latencies will be low and it will be easier to implement because no prediction needs to be done how the player has moved.

8. Appendices

8.1. Hardware description

The required machine performance is 256 MB of memory, a CPU of 1 GHz and a keyboard for ship control. Sound is supported by the system if available but not necessary for normal game play. The required software accessories are installed video drivers for OpenGL routines.

8.1.1. Minimal configuration

- 1.0 GHz CPU
- 256 MB primary memory
- Keyboard
- OpenGL drivers

This is a minimal configuration, which means anything lower that this may have a negative impact on the gaming experience.

8.1.2. Optimal configuration

- 2.0 GHz CPU
- 1024 MB primary memory
- ATI Mobility Radeon X700
- Keyboard
- OpenGL drivers

This is an optimal configuration, which means anything greater than this will do fine.