Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Distributed Computing

# Distributed Multiplayer Scenario

Bachelor Thesis

David Niggli

nigglid@student.ethz.ch

Distributed Computing Group
Computer Engineering and Networks Laboratory
ETH Zürich

**Supervisors:**
Michael König
Prof. Dr. Roger Wattenhofer

August 30, 2014

# Acknowledgements

I am grateful to my supervisor Michael König, who spent a lot of time during our weekly meetings to support me whenever I had difficulties. I would like to thank Professor Roger Wattenhofer for offering me the opportunity to write my bachelor's thesis at the Distributed Computing Group and learn so much about game development. Last but not least I thank my family and various friends for moral support.

# Abstract

The aim of this thesis is to design a multiplayer game including novel approaches considering the gameplay. One way to achieve this is to mix several already existing concepts, as well as to add innovative ideas in order to create a new user experience. The result is a real time strategy game in a first person view mode which takes place on a post-apocalyptic planet. To make the gameplay interesting, features must be balanced in order to find compromises between the different elder and well tested game concepts. My wish is that readers who want to develop games on a non-professional basis find this work helpful for knowing how to deal with challenges, be it in the architecture of the game engine or in concrete problems like implementing a three dimensional game map.

# Contents

**Acknowledgements** i

**Abstract** ii

**1 Introduction** 1

1.1 Defining "Distributed Multiplayer Scenario" . . . . . 1

1.2 Motivation . . . . . 1

1.3 Acronyms . . . . . 2

**2 Gameplay** 3

2.1 Mechanics, Dynamics, Aesthetics . . . . . 3

2.2 From Standard Mechanics To New Aesthetics, By Using A Mix And An Innovation . . . . . 3

2.3 An Obstacle To Overcome: The Lack Of Units . . . . . 4

2.4 Preventing Players From Having To Wait . . . . . 4

2.5 Game Concepts . . . . . 4

**3 Testing And Balancing** 7

3.1 Decorating The Map To Facilitate Navigation . . . . . 7

3.2 Resource Sharing . . . . . 7

3.3 Adjusting Attributes, Upgrades And Their Cost . . . . . 8

3.4 Shooting . . . . . 8

3.5 Flying Versus Jumping . . . . . 8

3.6 Speed Boost Versus Stamina . . . . . 9

3.7 Player Versus Building . . . . . 9

3.8 Changed And Dropped Buildings . . . . . 9

**4 Implementation** 10

4.1 Lobby . . . . . 10

4.2 User Interface . . . . . 11
4.3 Game engine . . . . . 11
4.4 Position, Orientation And Movement On A Sphere . . . . . 12
4.5 The Compass . . . . . 18
4.6 Random Looking Textures On Surfaces Without Tiling Effects . 20

**5 Analysis And Evaluation** **22**
5.1 Incomplete Features . . . . . 22
5.2 Conclusion . . . . . 23

**Bibliography** **24**

**A Appendix Chapter** **A-1**
A.1 Compiling The Code . . . . . A-1

Chapter 1

# Introduction

## 1.1 Defining "Distributed Multiplayer Scenario"

Computer game designer Chris Crawford defined a game as follows: It should be entertaining, interactive, as well as involve a goal to reach and adversaries who can interfere with each other's actions:
Watching a video for example is entertaining, but it is not interactive. An interactive plastic car for kids might be fun to play with, but it provides no particular goal to reach, so it is just a toy, not a challenge. Solving a Rubik's cube involves no other agents, so it's just a puzzle, not a conflict. When winning a marathon, one does not have to struggle with direct interference from adversaries, so it's just a competition, not a game.[1]

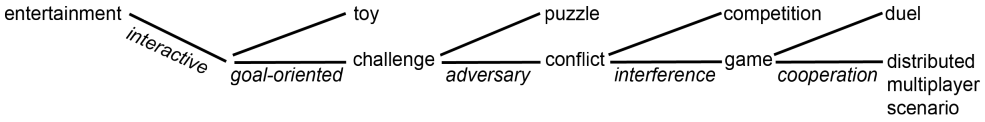


Figure 1.1: Distributed Multiplayer Scenario

As confrontations, competitions and games all imply exclusive success between groups of participants, they contain the notion of teams.
As Figure 1.1 illustrates, I would like to expand the previous definition in order to define a distributed multiplayer scenario, which is the title of my thesis: it should not only be competitive, but also cooperative:
In a chess game, the effort leading to the success or failure of a team is not distributed over several players but depends on only one individual, so it's just a duel, not a distributed multiplayer scenario.

## 1.2 Motivation

The video game industry has flourished within the last few decades. Pioneers have published games which were tremendously successful and the communities

around them have grown in huge numbers.
Games are further developed in order to improve the gameplay and keep up with technology, for example by making use of modern graphics hardware. New sequels are released and sold according to the reputation of predecessors.
Even though there are exceptions, most of the time distinct patterns are recognizable and differences between games of the same genre are minor.
This project is about innovation and trying to create something new, unseen so far. Although there might already exist games with similar ideas and features as those introduced in this project, the aim is to find innovative approaches and encourage readers to do the same.

## 1.3 Acronyms

RTS: Real Time Strategy
FPV: First Person View
FPS: First Person Shooter
MDA: Mechanics, Dynamics, Aesthetics[2]
TGUI: Texus' Graphical User Interface[4]
SFML: Simple and Fast Media Library[3]

Chapter 2

# Gameplay

## 2.1 Mechanics, Dynamics, Aesthetics

The goal of this project is to create a new user experience, thus the question is how to design the game. The MDA[2] model describes how game design and user experience relate to each other.

Mechanics describe the rules of the game and the control mechanisms defining various actions a participant can perform.

Dynamics describe the system that define how a participant can act in order to achieve his short- and long-term goals to ultimately win the game.

Aesthetics describe the resulting fun that the player experiences, which includes sensation, fantasy, narrative, challenge, fellowship, discovery, expression and submission.

## 2.2 From Standard Mechanics To New Aesthetics, By Using A Mix And An Innovation

As the aim of this project is to design a game with innovative approaches, the main focus will be on achieving new dynamics and ultimately new aesthetics, by using standard mechanics.
In this project, the mechanics are a mix of the rules found in typical RTS games and the control mechanisms found in typical FPS games.

Just as in a RTS game, players build structures in order to control certain areas of the map. Gathering resources allows expansion across the map and attack operations on the adversary.
Participants are units on the map and play in First Person View instead of having a global perspective, like in “Savage RX”. In fact there are no units on the map other than the players themselves.

Few video games actually take place on an spherical map due to the technical complexity; an exception for example is for example “Planetary Annihilation”. Some games use a torus-like map to achieve a similar effect while avoiding the complexity of the task, for example “Snake”. If you can’t make it, you can fake it...
This project rises to the challenge, implementing a three dimensional planet.

The hereby targeted core aesthetics are fellowship, challenge and sensation: players are encouraged to cooperate in large teams in order to achieve a bigger goal while experiencing the walk on a spherical map.

## 2.3 An Obstacle To Overcome: The Lack Of Units

As the players are the only units on the map, resources and buildings have to occupy the role usually attributed to units in a typical RTS game. One way of doing that is to make shots cost resources. That way to attack a lot, one needs a lot of resources, which would be the equivalent of many units. But those resources should be intended for attacking before the attack takes place, as units can not be used for anything else. Hence there is an attack resource equivalent to units. Vehicles are also a way to simultaneously encourage teamwork among players within a team, as well as replacing units. Unfortunately this feature is not yet implemented due to time constraints.

## 2.4 Preventing Players From Having To Wait

One of the main concerns when making the players be the units of a RTS game is what kind of activities they have to perform as units.
For example it is undesirable that they have to stay passively at one place while building, repairing or gathering. When players die in shooters, they have to wait for some time before they respawn. Instead of discouraging players from dying by such means, it is done differently: by making them lose all the acquired upgrades since their last death.

## 2.5 Game Concepts

#### Teams

Two teams face each other: North Pole versus South Pole.

### Resources

Each team has three types of resources. One for building, one for attacking and one for upgrading. Players of the same team share all resources.

### Players

Players have attributes like a life points cap and regeneration value, a stamina cap and regeneration value, an attack power and range. When all life points are lost, the player dies and respawns on the pole of his team. Stamina is required to sprint or gain altitude. Players can shoot laser beams with a certain attack power and range, which cost the team resources.

### Buildings

Teams expand their territory and control certain areas of the map by building the following structures:

- The *power plant* is essential, as it is allowing the construction and activity of other buildings around it within a certain range.
- The *g*atherer can only be placed on resource occurrences and gathers at a certain rate.
- The *s*hooter is a tower that shoots on enemy units within a certain range, thus draining their life points.
- The *h*ealer is a building that heals wounded players of its team, again within a certain range.
- The *s*peeder is a structure that allows players of its team to travel faster through controlled territory by increasing their stamina, hence allowing constant sprinting.

### Upgrades

Each building can be improved or improve the player by buying upgrades. Depending on the building, they increase the building's own range, performance or life points cap and the player's range, performance, or cap in a specific domain.

For example a *s*hooter can upgrade their own life points cap, their own shooting range, their own shooting power or the players' shooting range, the player's shooting power. Along with the upgrades, there is an option to repair a damaged building, which is more advantageous then rebuilding one, if the building has been upgraded.

### Game Object

A team starts with a *power plant* on one of the poles.
The game object is to build structures in order to gather resources permitting various upgrades that enable more efficient exploration, expansion and destruction of adversary structures. The game is won when all adversary *power plants* are destroyed.

Chapter 3

# Testing And Balancing

## 3.1 Decorating The Map To Facilitate Navigation

On a map without corners, orientation is a main challenge. The compass gadget, although it can guide a player directly to the own or enemy pole, is not enough help for a beginner and still a limited one for advanced players.
In order to give players an idea of where they are, obstacle types such as black rocks, gray rocks, volcanoes, monoliths, walls (and more could be added in the future) are used to define regions of the map.
One side of the map is dominated by gray obstacles and the other half by black ones. In their middles reside respectively a bunch of monoliths and an elliptic volcano chain. Resources are to be found in both of these places, allowing a larger sight by climbing on a monolith, or natural protection against adversaries through life point draining volcanoes.
As northern and southern hemispheres are references with an absolute location whereas West and East are only relative, a fragmented wall divides the planet from pole to pole in a straight line in order to provide an equivalent reference. Along that wall can be found four further major resource occurrences, two in each hemisphere.
In case the players have any doubt about where to head because the black and gray rock occurrences are insufficient to give them an idea of what bigger entities may be found in that region, there are sign posts at both poles that indicate where the volcanoes and monoliths are.
Few resource occurrences are spread out randomly across the map.

## 3.2 Resource Sharing

In order to prevent players from using all the upgrade resources on themselves, costs grow polynomially, thus leaving other players from the team the time to upgrade themselves to approximately the same level. Unequal resource use should only happen if intended as a strategy and agreed upon amongst team mates.s

achieved.

## 3.3 Adjusting Attributes, Upgrades And Their Cost

Exponential growth was experimented with but abandoned because of too extreme consequences; it was replaced with attribute upgrades that increase performance linearly, and costs that are polynomial in the level.

## 3.4 Shooting

### Discrete Versus Continuous Shots

At first the idea was that players shoot bullets. While implementing the feature, the first prototypes were just performing a ray trace every frame when shooting in order to determine the target, and the idea of continuous laser beams remained as it fit into the theme of the game. A resulting issue to be addressed in future is for example that lag will cost amounts of resources spent on unwanted shots.

### Resources Versus Munition

Along with life points and stamina came the idea of a munition bar, which would only be refilled when a player comes near one of his buildings or maybe a specific building. The fact that shots cost resources makes players not run out of munition unless they can't afford it anyway. This feature was dropped due to exaggeration in realism and no real improvement on the gameplay.

### Camera Origin Versus Shot Origin

When the player's own shots were displayed, they covered the screen. To prevent that, the camera position and the shooting source have been separated. Implementation required some computation on client side to translate the visual target into a corresponding shot direction.

## 3.5 Flying Versus Jumping

In the early stage of the development, the player could gain altitude quite easily. The thought occurred to include and encourage flying as a feature. But after testing it, the decision was made to weaken the ability to gain altitude to prevent its abuse. Another issue is how much sideways movement control a player has

when he is “in the air”. As the theme is post-apocalyptic, it was decided that players are robots and given the same level of movement control at every altitude.

## 3.6 Speed Boost Versus Stamina

*S*peeders are supposed to enable fast travel in controlled territory. Testing the speed boost led to the observation that the movement control decreases. This issue is solved by restoring stamina to the player instead of directly increasing his speed, hence allowing him to move faster by sprinting when he decides to.

## 3.7 Player Versus Building

As buildings play an important role in the game, it is important not to give the player too much power. On the other hand the player is still the user, and buildings should not be too powerful either. While balancing the attributes and testing the gameplay, both extremes were experienced: from players destroying anything in their way in the blink of an eye to buildings standing ground and taking over the defence of occupied territory with overpowered towers. A compromise between the two situations was established by empowering and weakening each side back and forth until a healthy balance wa

## 3.8 Changed And Dropped Buildings

In the beginning all buildings had the same dimensions. For better recognition they were changed to have different shapes and sizes.
The healer towers heals players of its team within a certain range of action; the speeder tower was replaced with a highway tile, which restores stamina by some technological means when players of its team walk on it. It was planned to make entire roads with low cost, in order to create highways within the controlled territory. This feature requires path finding and automated placing of individual tiles; due to time constraints the feature is not completed.

Along with the speeder came the idea of a slower tower that would slow down adversaries in order to give the team more time to react to attacks. This has been left out because the first person view is already a handicap for map overview and slowing down players even more does not make much sense.

Chapter 4

# Implementation

## 4.1 Lobby

When the application is launched, the lobby is entered, as illustrated in Figure 4.1. On the left are a text box to enter a user name, a list box containing all games hosted in the local network and buttons to exit the application, refresh the list of games or join a selected game. Additionally there is a text box to enter a game name, as well as a button to host a game under that name.

When hosting or after having joined a game, the right side displays a list of all joined players along with some attributes, like the selected teams or whether they are ready to start the game or not. A player can select a team with a combo box and get ready to start the game with a check box, which is unchecked automatically when settings change, for example if a new player enters or someone turns to an other team. Additionally there is a chat box and a text box with a send button for players to communicate while the game is set up.
When all players are ready the game is launched. When the host exits or the game terminates the lobby is entered again.

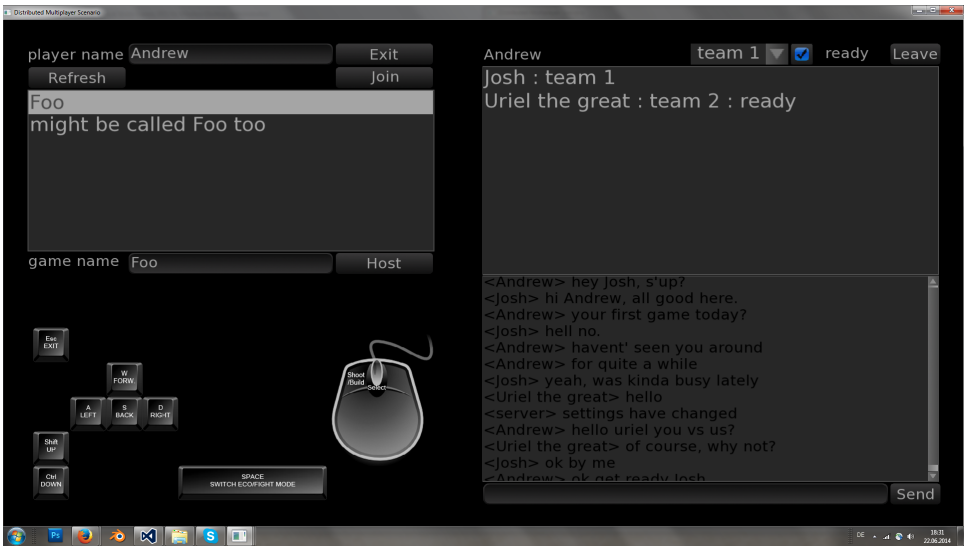


Figure 4.1: The lobby

## 4.2 User Interface

The control mechanisms are pretty much those of a typical FPS game: a player moves using WASD keys, turns himself using the mouse, gains altitude with the space key and performs actions using left click.

There are two modes of action: an economic mode in which a player can place new or upgrade existing structures and a military mode in which a player can shoot. These actions are performed using left click. A player can toggle between the two modes using the E key; right clicking also brings the player back to military mode.
In economy mode, when a player targets the ground, a building appears, if it can't be built due to collision, distance or lack of resources, it is half transparent. The different building types can be iterated through by scrolling or directly accessed using the corresponding number key on the keyboard, and built by left clicking. When a player targets a building he can apply upgrades on it, which can also be scrolled through or accessed directly with the corresponding number key.

## 4.3 Game engine

### Framework

The following C++ libraries were used in this project:

- SFML[3] provides five modules, which allow threading, networking, audio, windowing and 2D graphics.
- TGUI[4] provides high-level GUI elements building up on SFML.
- OpenGl[7] was used for 3D scene rendering.
- Bullet Physics[5] was used for real time physics simulation.

#### Central Server

The network is centralised. The server is the ultimate reference considering the game state. This avoids complicated protocols when it comes to the synchronization of the virtual reality. On the other hand this architecture implies that the server is the bottleneck concerning network latency. Client side movement predictions are a possibility to overcome the issue, but in this project there was no need for it, as the message complexity was very limited.
The clients receive user inputs, translate them in to local consequences such as camera movement or building visualization before the user clicks to build, and forwards game state relevant information to the server.
The server receives user inputs from the clients and applies the action to the game state which he increments every frame, before broadcasting the changes to all clients. The clients receive updates and change the data accordingly, which is then displayed on the screen.

## 4.4 Position, Orientation And Movement On A Sphere

A player's position and orientation (how he is rotated in the world) have to be kept track of and updated according to movements, as illustrated in Figure 4.2. The sphere is centred in the origin of the coordinate system, hence the y-axis intersects it in an upper and a lower pole.

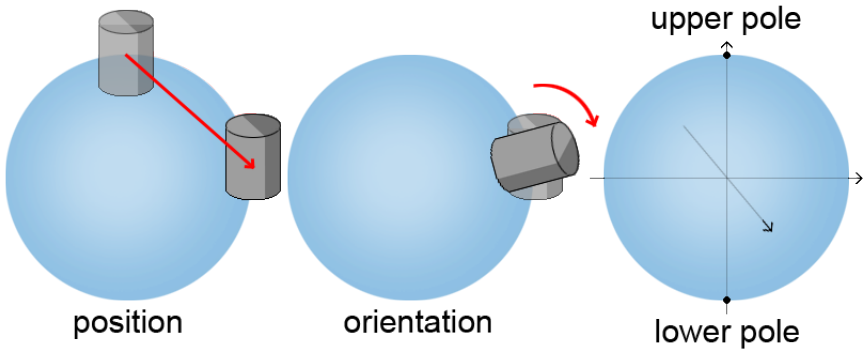


Figure 4.2: Position, orientation, poles

### Spherical Coordinates

In order to represent a player's position, a first approach was to work with spherical coordinates:
Instead of using three coordinates x, y and z for each dimension, two angles theta and phi and a distance r to the origin would be used.

An advantage of this representation model is that theta, phi and r directly correspond to the notions of latitude, longitude and altitude, as illustrated in Figure 4.3. For example, jumping just affects one coordinate (the altitude) independently from the other two.

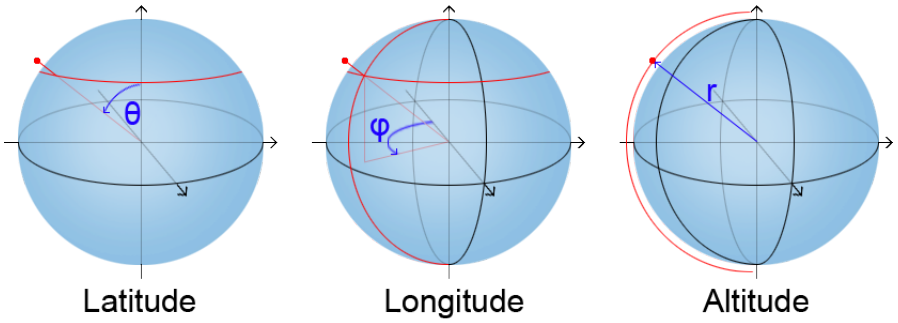


Figure 4.3: Spherical coordinates

A disadvantage of spherical coordinates is that it can involve case distinctions between the hemispheres: when an angle has a value range from zero to two pi, trigonometric equations can have multiple solutions.

### Transformation Matrices

OpenGL as well as Bullet Physics[5] use matrices to describe both position and orientation of entities. These matrices have to be computed anyway for the rendering process and the physical simulation.
The state of an entity is described by a vector that is its position, and a quaternion representing its orientation in the world, which is defined by a rotation axis and a rotation angle, as described in Figure 4.3. The Bullet Physics[5] API and the GL Mathematics library both provide the infrastructure to build matrices from these components and vice versa extract the position and the orientation from the matrix.

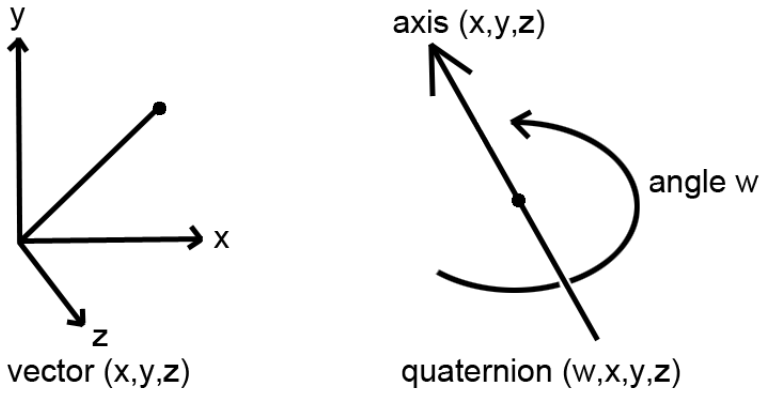


Figure 4.4: Vector and quaternion

### From Movement Relative To The Player To Actual Movement On The Sphere

The player orients himself by using the mouse and moves by pressing keys on the keyboard.
Those inputs are interpreted relatively to the player's reference frame. For example a player can jump while moving forward and rightward at the same time. That means that relatively to his current orientation, he wants to move along a vector going up, forward and rightward.
That vector is then rotated and translated in the same way as the player in order to obtain the final movement vector, as illustrated in Figure 4.5.

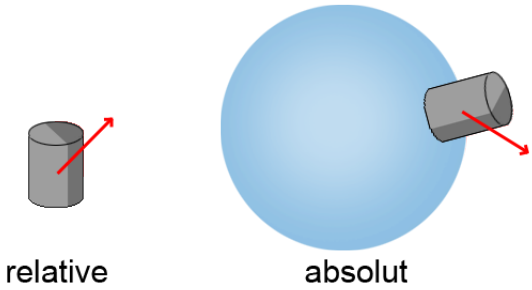


Figure 4.5: From movement relative to the player to actual movement on the sphere

### Keeping The Player Upright Despite The Laws Of Physics

Using a real-time physics engine is nice for recovering from collisions or using its built-in ray-trace feature, but some of its consequences are undesirable: in real life, a person - or any object for that matter - can fall to the ground; in a video game, a player's character shouldn't stumble in the first place.
Because that is a common issue, the physics engine allows to manipulate and restrict the physical properties of physical entities.
On a plane, the engine can be allowed to only affect an object's position, and forbidden to affect its orientation, which therefore remains unchanged over time, no matter what events occur; hence guaranteeing that the player will always be upright.
On the sphere, leaving a player's orientation unchanged is not wanted, as the orientation of a player changes depending on his location. But one can think of using a similar tactic: the player can be reoriented manually to guarantee that he is standing upright.
To do that, the quaternion that transforms the position vector of the upper pole into the position vector of the entity can be computed as illustrated in Figure 4.6: the rotation angle is the inverse cosine of the dotproduct of the two normalized vectors, and the rotation axis is their crossproduct.

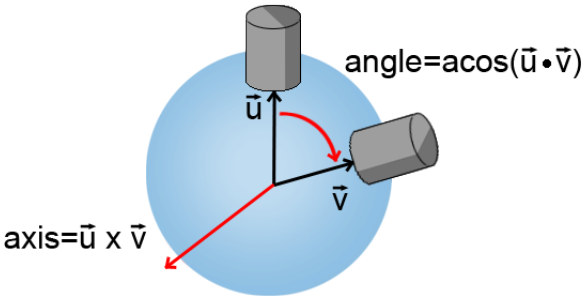


Figure 4.6: Reorientation quaternion

That rotation can be applied to the entity, hence orienting it up right. That way, there is an injective projection from the entity's initial orientation on the upper pole to that on any location on the sphere.

**Observation and Analysis: Strange Movement Behaviours Due To Course Deviation**

After the implementation, walking was tested on the sphere. At first it seemed to work allright, but soon issues were discovered. The further away a player walked from the upper pole, the more movements started behaving strangely, following a particular pattern: when walking nearby the lower pole, the player got trapped in a circular movement path around it while moving just forward. On the upper hemisphere, the deviations from the course were negligible, in fact they were not even noticeable, which led to the erroneous hypothesis of the need for some kind of case distinction between hemispheres.
In order to understand what was exactly happening, the physics engine's simulation was removed in order to avoid any black box. The three steps that should happen in a frame were manually implemented: First the player should move, then gravity should pull him down to the ground, and finally the player should get reoriented to be upright. Big steps per frame one after the other are debugged, in order to observe the scene after each step. The findings were that after a step, a player was no longer facing the same direction, but had made a turn, as illustrated in Figure 4.7.

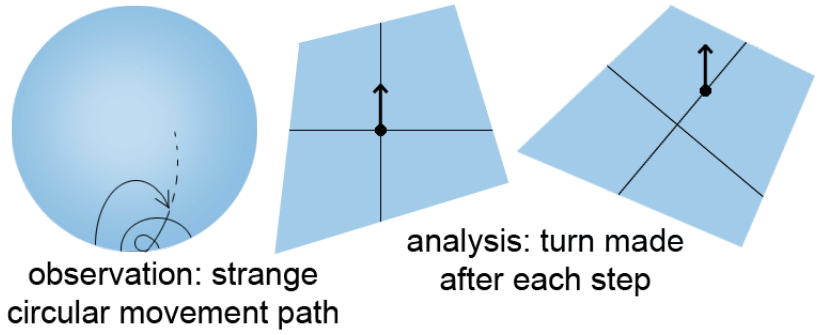


Figure 4.7: Observation and analysis

The algorithm reoriented an entity by projecting it from the top to its new location from scratch considering just its position like on a plane. For two neighboring points on the lower hemisphere, that projection did not result in similar rotations of an object around itself, hence the course deviation.

### A Sphere Is Not A Curved Plane

The mistake was to just consider the sphere as nothing more than a curved plane. This approximation was locally good enough on the upper pole, but when getting closer to the other side of the sphere, it held less and less. In fact two opposite directions going away from each other on the upper pole meet again on the lower pole.
It turns out that walking on a sphere is not quite the same as walking on a plane. Let's first illustrate that with the example illustrated in Figure 4.8 and then generalise it to a more global principle:
Standing on the North Pole and facing a certain direction a player can go down the sphere until he reaches the equator; after moving half way around the sphere he finally walks back up to the pole. He finds himself on the same position as he started, but facing the exact opposite direction, hence he just passively rotated by a hundred and eighty degrees. More generally, any given orientation can be achieved on any given location on the sphere by just walking on it without ever actively rotating.

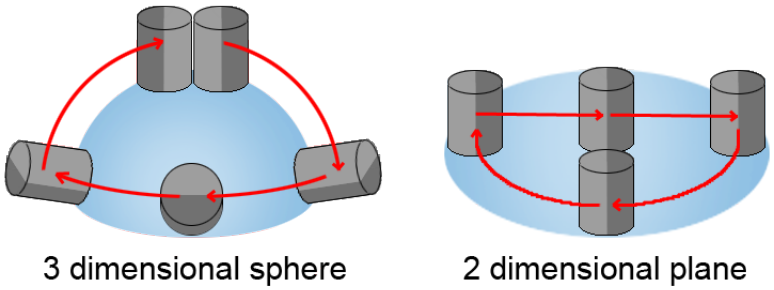


Figure 4.8: Walking on a sphere

#### Active And Passive Rotation

The findings were that the orientation of an entity does not only result from the history of active rotations decided by the player through mouse movements but also from the history of passive rotations resulting from movements on the sphere.

#### A Constructive Approach

The problem is solved by computing the new orientation constructively according to the last one, instead of using the initial one to do it from scratch.
To do so, the last position and orientation are stored before running the simulation and are then used to compute the new orientation.

#### A Polished Solution

One problem remaining is that computation errors accumulate over time and never get corrected. The more elegant way of doing it, is to compute the side effect rotation caused by the movement and add it to a "pitch" rotation resulting from both active and passive rotation. After that, one can recompute the orientation frame by frame, by first applying the pitch and then the projection. Accumulated negligible errors on the pitch are not noticed, as the mouse control precision is also limited. On the other hand cumulative yaw and roll deviations are preferably avoided.

## 4.5 The Compass

One orientation feature is the compass gadget which always points toward the upper pole. This is achieved by projecting the pole onto the player's relative two

dimensional plane and computing the angle between the forward vector and the pole's projected position vector, which is defined uniquely by the up vector. The compass is then rotated by that angle, as illustrated in Figure 4.9.

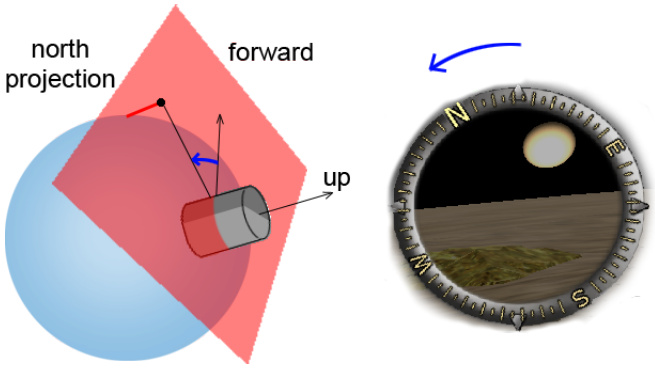


Figure 4.9: North Pole projection

### Projecting The Pole

Let n be the unit vector normal on the plane; n is equal to the normalized up vector.
Let d be the distance between the pole and its projection on the plane; d is equal to the dotproduct of the pole's position relatively to the player and n.
To compute the projected position of the pole, d times n is added to the pole's position.

### Compute The Angle

In the method described in Figure 4.5 the rotation angle of a quaternion is computed using the inverse cosine of the dotproduct of the two normalized vectors, which has two solutions but returns the one in the range between zero and pi.
Because the crossproduct and the dotproduct are complementary, the sign of the angle is taken care of by the direction of the rotation axis, hence together they define a unique rotation, as illustrated in Figure 4.10.
However the rotation angle of the compass has a range going from $-\pi$ to $\pi$. The up vector is used to define a unique solution:

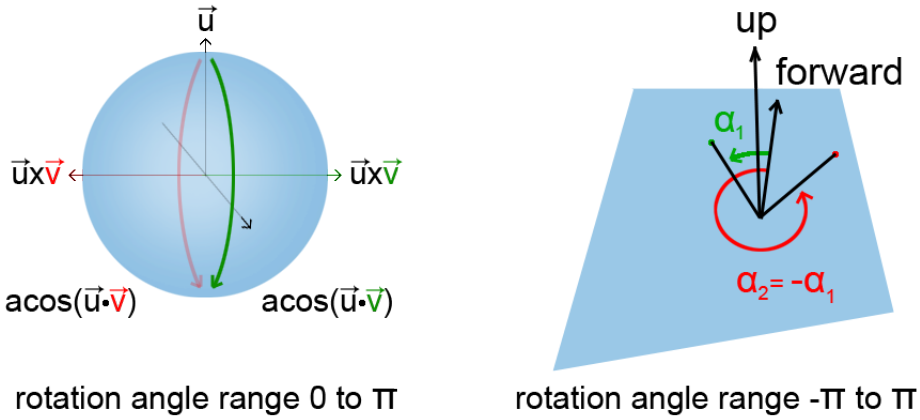


Figure 4.10: rotation angle uniqueness

The crossproduct of the forward vector and the up vector results in the right vector.
If the dotproduct of the right vector and the projected position is positive, the angle between them is acute, otherwise it is obtuse.
If the angle between the right vector and the projected position of the pole is acute, the pole is to the right of the player, otherwise it is to the left.
If the pole is to the left of the player, the angle is positive, otherwise it is negative.
The angle has now a value range between $-\pi$ and $\pi$.

## 4.6 Random Looking Textures On Surfaces Without Tiling Effects

This part was experimented with in the plane; after the decision was made to implement a spherical map it was dropped.
The goal was to construct big random looking surfaces with few texture variations. The plan was to use randomness to eliminate the tiling effect. Square tile textures have borders B1 to B4 (top, bottom, right, left). Each of those borders has an interface I1 or I2. Hence there are $2^4$ tile textures with all possible interfaces at all possible borders.
Each of those textures is represented by an integer that is 1 in the beginning and is then multiplied (or not) by prime numbers 2 through 7 if the borders B1 to B4 have interface 1 (or I2).
A function was programmed, that generates an array of integers that represent which of the 16 textures is to be used for each tile. It looks up the zero to two neighbouring tiles already determined previously and randomly chooses a texture amongst the 4 to 16 compatible ones. A mesh is then generated using those textures.

Along the texture map, a randomized heightmap can be used to generate relief with arbitrary constraints on slope and boundaries.

Chapter 5

# Analysis And Evaluation

## 5.1 Incomplete Features

Due to time constraints, some of the planned features were not completed. They give an idea of how existing features can be improved:

- When a player places a highway tile on the map, a set of tiles starting at the closest already existing building and ending at the tile closest to the targeted position is computed. The cost depends on the number of required tiles. This improves the visual aesthetics as well as the gameplay as the construction of continuous roads is automated and permits constant sprinting.
- Power boosting vehicles that can only be deployed by several participants encourage cooperation amongst players within the same team. For example one person can only drive, but much faster than without the vehicle, and an other one can only shoot, but with more damage points than without the vehicle.
- To make the map less monotone, the planet consists of an amount and variation of spheres that is small and random. The texture of its surface are randomized in order to remove the tile effect.
- Instead of strictly splitting the purpose of resource types, some combinations make the balancing more interesting.
- the lobby consists of different tabs: one for settings, one for the list of local games, one for the currently joined game etc.
- a master server creates server threads listening on different ports. The list of games displays attributes describing the number of players, if the server runs on the master server or not etc.
- The following is more of a joke that crossed the mind: settings allow the client to set a theme, for example in the fast food theme the monoliths are

French fries, the resources are ketchup, mustard and mayonnaise, gatherers are grills, a doughnut surrounds the planet on the equator etc.

## 5.2 Conclusion

The release of a video game involves big teams that work full time on it during several months or years. Without having previously ever worked on a bigger project, after a few months I have learned a lot about the used frameworks and game development in general. Had I to reimplement it, the structure of the engine would look differently. Not all the planned features were completed, but the general idea of an RTS-like game taking place on a sphere struck me and I am planning to take the idea further.

# Bibliography

[1] by Michael Stevens' youtube channel "Vsauce": *why do we play games?*
http://www.youtube.com/watch?v=e5jDspIC4hY

[2] by Robin Hunicke, Marc LeBlanc and Robert Zubek: *A Formal Approach to Game Design and Game Research*
http://www.cs.northwestern.edu/ hunicke/MDA.pdf

[3] SFML: *Simple and Fast Media Library*
http://www.sfml-dev.org

[4] TGUI: *Texus' Graphical User Interface("GUI library for SFML")*
http://www.tgui.eu

[5] Bullet Physics: *Real time physics simulation*
http://www.bulletphysics.org

[6] GitHub: *Git repository*
http://www.github.com

[7] OpenGL: *Open Graphics Library*
http://www.opengl.org

Appendix A

# Appendix Chapter

## A.1 Compiling The Code

The external libraries mentioned in chapter "Implementation", section "Framework" can be downloaded for free on GitHub[6]. They need to be compiled and linked to the source code of this project, which should be cross platform (Windows, OS X and Linux) but was only tested on windows compiled with visual studio 2013.