TEAM A5

REPORT

ALEXANDRE PORTUGAL - (1231727)
BELLA DUNOVSKA - (1250580)
MARTIN MIHOV - (1229174)
SAMUEL HILL - (1240663)
TENDAISHE NYAMAPFENI - (1251414)

1. INTRODUCTION

Through this report we hope to provide a clear representation of our development process as a team, significant detail regarding implementation specifics and to thoroughly document the evolution of our concept outlined in the System Requirements Specification, for our game Sknat.

Our Team have developed a new self-contained game of collaborative and competitive nature as specified in our SRS. The game itself is a 2d arena game where players score through killing each other and completing objectives, each player controls a tank. The gameplay revolves around killing the opposition, obtaining the mutual objective (key) and holding it in the middle territory of the map for 3 seconds. Completion of this action causes a missile to be unlocked and subsequently launched, destroying the opposition's base. Completion of the objective or death of the key holder causes the key to respawn randomly in a new location. The gameplay is made expansive through obstacles, projectiles (of limited range) and a combination of scoring mechanisms, allowing players to pursue different strategies.

As per the requirements highlighted to us by Manfred and Ella, we felt strongly compelled to incorporate real-time networking to promote collaboration and competitiveness. This makes our game fast-paced, complex and more engaging. The game is intended to be team-based with two players per team, but as the project progressed we extended ourselves to further game-modes. This allows the different players to experience a unique perspective on their respective machines. We have also been able to incorporate some extensions earlier mentioned in the specification. The game has been extended to include a lobby text-based chat, with other pre-game customizations, and a competitive AI allowing us to provide a 1v1 trial mode, and a co-op mode, allowing two players to collaborate solely against AI. The game has a graphical interface with panels for menu navigation and the map itself. The menu systems provide an easy means for users to navigate to their desired game-mode, and choose to host or join games, leading them to the pre-game lobby state. From here players may communicate using nicknames, switch teams, and must indicate when they are ready for the game to begin. Once this has happened the host may start the game.

The game has various single player and multi-player modes, some of which contain AI.

Single Player

- **1vs1**: Human vs single AI.

Multi-player

- **1vs1**: Player vs Player
- **2vs2**: 2 Players vs 2 Players
- **Co-op**: 2 Players vs 2 AI Players

Our report will detail the stages involved in developing our game and our decision-making behind major implementation choices. We will provide a detailed class structure and top down design, clearly demonstrating the modular nature of our approach. HCI has been a major consideration of ours and implementation choices have always been made with this in mind. This is reflected through our back-end implementation decisions through to our graphical interfaces. From choice of network protocols, client-side vs server-side processing, and colour schemes. We will include all details of our HCI and general game design. As previously mentioned, our game incorporates real-time networking. Therefore networking became a complex and crucial component of our game causing crucial implementation details to be made early on, shaping our progress throughout the 10 weeks. We will

highlight in the report how we achieved our real-time networking, the protocols we used for instantiating clients, pre-game lobby state and synchronising data during the game-loop itself. We will discuss our considerations of Software Engineering principles which we used to optimize our development and minimize risk. We will then evaluate our performance as a team, and the product we developed, as well as reflect upon our organisation and systems used for group cohesion.

2. SOFTWARE DESIGN

2.1. CLASSES

2.1.1. AI

- **ArtificialTank** - Implements an AI behaviour for a Tank object. Simulates events and sends them to the server based on the current game state.
- **AStarSearch** - Does an A* search between two points.
- **Node** - A step in the search, containing its parent and the current action.
- **Point** - A point with x and y.
- **Search** - Interface for providing functionality of any search

2.1.2. Client

- **ClientListener** - Listens to the server and decodes the string it receives
- **GameClient** - Responsible for the interaction between local game data and the data sent by the server.

2.1.3. Game

- **GameData** - Contains all the data necessary to run the game.
- **GamePanel** - Panel where the game is drawn to. Also captures and processes user input.
- **SoundPlay** - Class responsible for playing sound.

2.1.4. Game.objects

- **Entity** - Base class for all entities in the game.
- **FloorTile** - Base class for map tiles which each floor tile type extend.
- **Obstacle** - Base class for map tiles which each obstacle tile type extend.
- **Key** - Class that represents a key.
- **Missile** - Class that represents a Missile. Extends projectiles and has methods related to how it moves.
- **Objective** - Class that represents an objective.
- **Projectile** - Class that represents a projectile. Has methods to calculate projectile motion.
- **Tank** - Class that represents a tank. Has methods to calculate pointing angle based on mouse coordinates.

2.1.5. Map

- **Maps** - Class that generates a map for our game.

2.1.6. Menu

- **EndGamePanel** - Class responsible for showing victory, defeat and draw screens.
- **GameModePanel** - Class responsible for showing the screen that lists all game modes for multiplayer.
- **LobbyPanel** - Class responsible for showing the game lobby.
- **MainMenu** - Class where all different panels initialized and setup.
- **MultiplayerPanel -** Class responsible for showing the screen that allows you to join or create a session.
- **StartMenuPanel** - Class responsible for showing the screen that allows you to choose multiplayer or single player.

- **TeamTableModel** -Renders based on changes to the tanks game data. Returns values for tank ID, name and ready status.
- **TeamTableRenderer** - Performs validation checks on team changes and colours table accordingly , indicates corresponding player

2.1.7. Networking

- **Client2Server** - A parser used by the client to decode events received from the server and to encode events to be sent to the server.
- **Server2Client** - A parser used by the server to decode events received from the clients and to encode events that need to be sent to the clients.
- **TCPClient** - A class for creating a TCP/IP client that sends and receives packets to and from the server.
- **TCPConnection** - A class for creating a TCP/IP server that sends and receives packets between the players. Creates a sending and receiving threads for the new players.
- **TCPServer** - An abstract class for creating a TCP/IP connection.

2.1.8. Server

- **GameServer** - Class responsible for establishing the connection with the clients, for providing the communication between the clients, updating the game state, based on the current state and making sure every client has the newest game state.

2.2. UML DIAGRAM

Please refer to appendix 1.

3. GAME DESIGN AND HCI

3.1. GAME ENGINE

The game is run using a **jPanel** that listens for key presses and mouse actions and renders and draws the game data that is store in the **GameData** class. The rendering methods are coded in a way that can be configured in order to render it more or less often. This allows the game to feel as smooth as possible regardless of what computer is run on.

The game data class contains all game related entities that is needed to run the game. This includes obstacles, tanks, floor tiles, projectiles, etc… These are all being constantly updated by the **GameClient** class, but all these updates will be explained in the networking section. This game data is being constantly drawn on the panel.

Depending on the entity, there are several different steps that need to be done. Let's take tanks as an example, which have two parts that need to be drawn: the body (referred to as tank base) and the cannon on top (referred to as tank top).

As mentioned in the HCI section, the tank top needs to be pointing at the mouse pointer whilst the tank base needs to be pointing at the direction that the tank is moving in. Thus we must rotate the tank's images in order to draw them pointing at the right directions. In order to do this we must know where the mouse is, which is achieved with a simple call from the **MouseInfo** class, and what keys are being pressed. To find what keyboard.

Movement control of the user's tank is done client side. There is a thread constantly checking if there are buttons clicked. If a specific button (or buttons) (w, a, s or d) are being pressed, it moves the tank in the direction that the combination of buttons requires. It then rotates the tank base side to face that direction. Here is an example of where it interprets the buttons pressed and finds the angle. In this case it is testing for pointing northeast or southwest, which is the same since our tank's base are symmetrical both horizontally and vertically.

```
// Checks which angle it should be pointing at
// Pointing towards the top right (or bottom left)
    if ((wPressed == 1 & dPressed == 1 & sPressed == 0 & aPressed == 0) | (sPressed ==
    1 & aPressed == 1 & wPressed == 0 & Pressed == 0)) {
        gameData.getTanks()[myID].setBotAngle(Math.PI / 4);
    }
```

3.2. HCI

3.2.1. Lobby

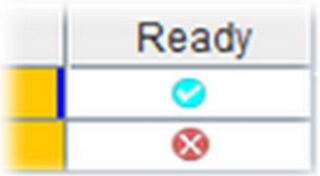


Figure 1: HCI - Ready or not ready

Ready or Not Ready
Figure 9 Shows the visual indicators which show which player is ready and which player isn't.

Invalid Team Switch
When a player tries to switch team and there is not enough space in the opposite team; their name will be highlighted red to indicate an invalid operation, as shown in figure 10.

Figure 2: HCI - Invalid Team Switch

3.2.2. Gameplay

In order to provide good HCI during gameplay we did quite a number of things. The first being the choice of colours. Our initial colours where green and red for the teams but later on we discovered that this was not very user friendly in terms of colour-blind users. So after running some tests on an online vision simulator we decided to use blue and red because in most cases they were the most distinguishable colours.

The use of these colours goes deeper into the gameplay. During testing we realised that when playing the game you tend to focus more on health of the tank instead of the actual tank, so we also decided to use colours which are friendly to colour-blind players. Red for enemies because it is intuitive, blue for allies and yellow to identify yourself. Below is some of the other HCI elements we implemented in the game.

Figure 3: HCI - Winning Team

Winning Team
During gameplay the game timer will take the colour of the winning team. This feature allows players to identify who is winning without losing consecration on the action because it is a visual cue which doesn't require thinking unlike reading.

Figure 4: HCI - Objective Countdown Timer

Objective Countdown Timer
Also during gameplay, when a player has the key and goes on top the objective territory a visual timer appears and starts to count down; as shown in Figure 12. This is very intuitive and it also tells the player how long they have to wait to get the points.

Aiming
During gameplay the player is able to shoot his tanks cannon at his mouses position by clicking the left mouse button. To make this more intuitive we made it so that the cannons top points at where the mouse is pointing.

Window Resizing
Due to concerns regarding screen size raised by Manfred in one of his presentations we made it so that our game's graphics scale based on the size of the window. This allows our game to be playing on any sized screen and leads to a much more pleasant user experience.

4. GUI

4.1. GAME FLOW

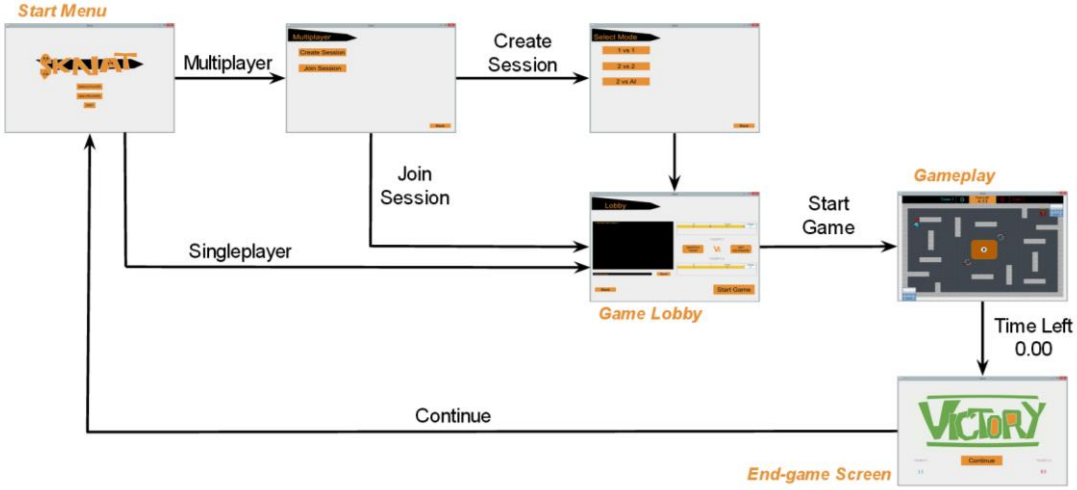


Figure 5: Game Flow Design

The diagram above shows how the game flows from one screen to another. The game always starts on the 'Start Menu' screen were you get 2 options; single-player and multiplayer. If you select single-player you will be taken straight to the game lobby and you will be able to start the game. However the multiplayer route has more options. If you select multiplayer you need to first decide whether you are creating a session or joining a session. If you are creating session you will automatically become the host and you can choose 3 game modes; 1v1, 2v2 and Coop/2vAI. Once a session is created you will moved to the lobby. To join an existing session you need to input the IP address of the host and if successful you will be moved to the game lobby.

The game lobby contains a chat interface which allows players communicate with one another. Each player in the lobby can set a custom nickname and can switch to the opposite team if there is space. The game can only be started if all the players are ready and there are icons which indicate whether a player is ready or not ready, so if someone is taking long you can have a word with him/her in the chat.

Once all players are ready the game can be started and all players will be moved to the gameplay screen. This is where all the action happens. Players will fight each other until the game timer reaches 0. The team with the most points will win and there is a tie the endgame screen will change accordingly and of course there is also a defeat screen. From the end game screen you can only go back to the Start Menu where the game loop begins anew.

4.2. MENUS

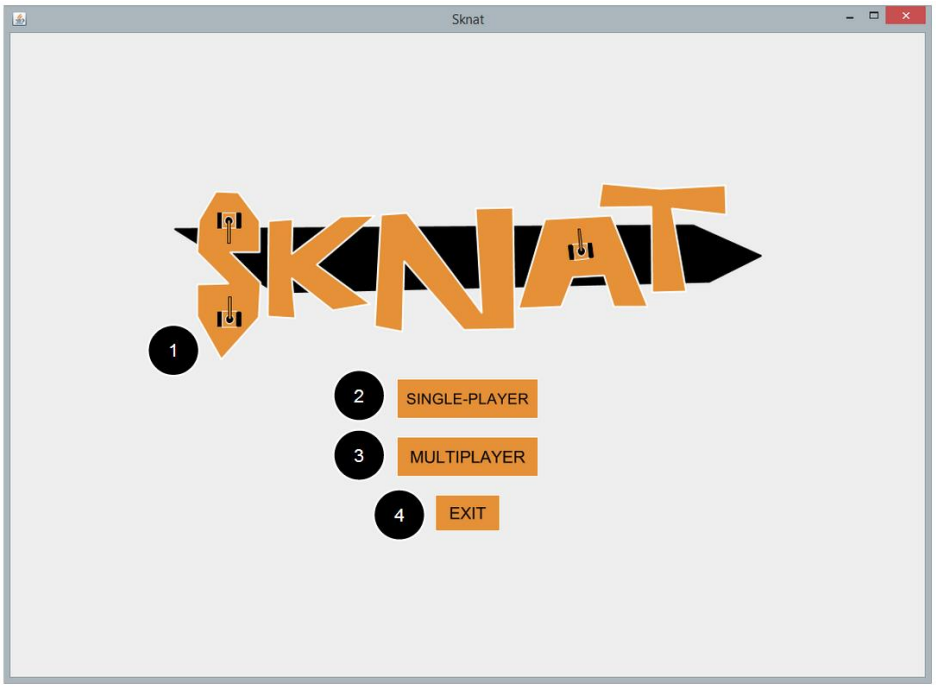


Figure 6: Start Menu

Object Number	Name	Description
1	Logo	This is the game logo.
2	Single-player	This button allows the player to start a game against the AI.
3	Multiplayer	This button allows the player to participate in a multiplayer game.
4	Exit	This button will allow the player to exit game.

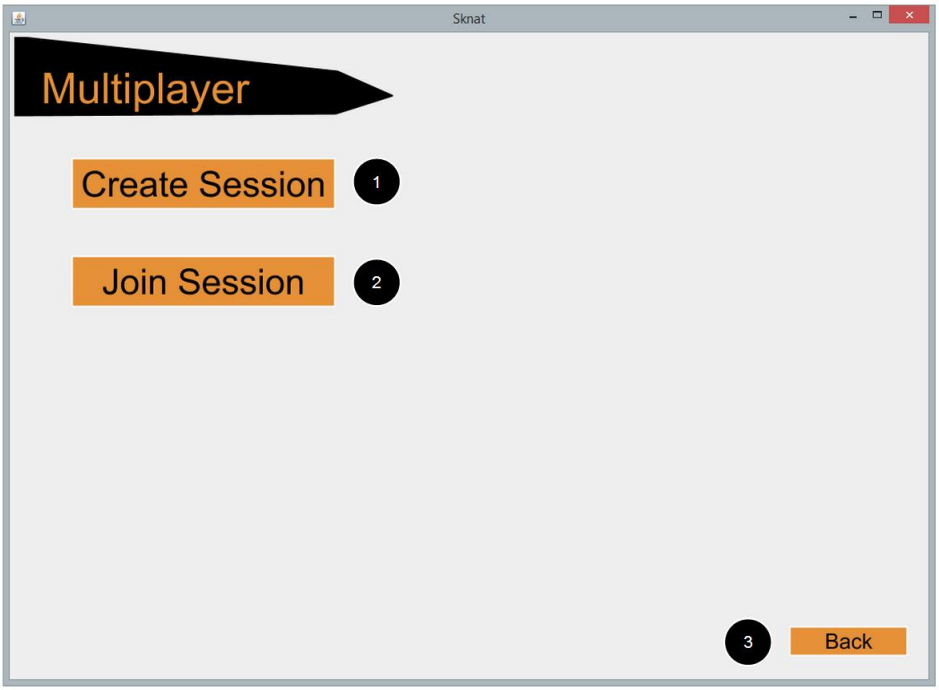


Figure 7: Multiplayer

Object Number	Name	Description
1	Create Session	This button will allow the player to create a new game session.
2	Join Session	This button will allow the player to join an existing game session.
3	Back	This button will allow the player to go back to the Start Menu.

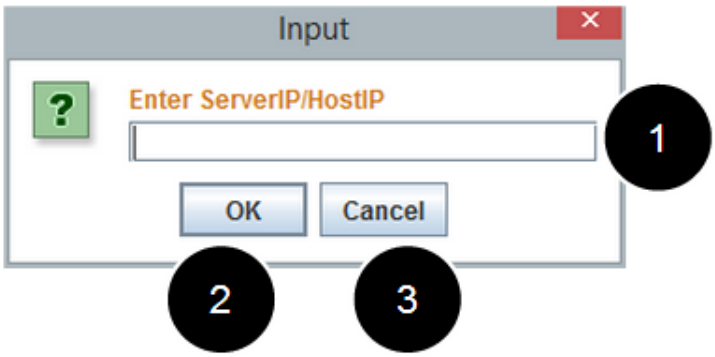


Figure 8: Multiplayer - Join Session

Object Number	Name	Description
1	Host IP	This text box will take input from the player. The input has to be the IP address of the Host.
2	OK	This button will allow the player to join an existing session.
3	Cancel	This button will allow the player to cancel the operation.

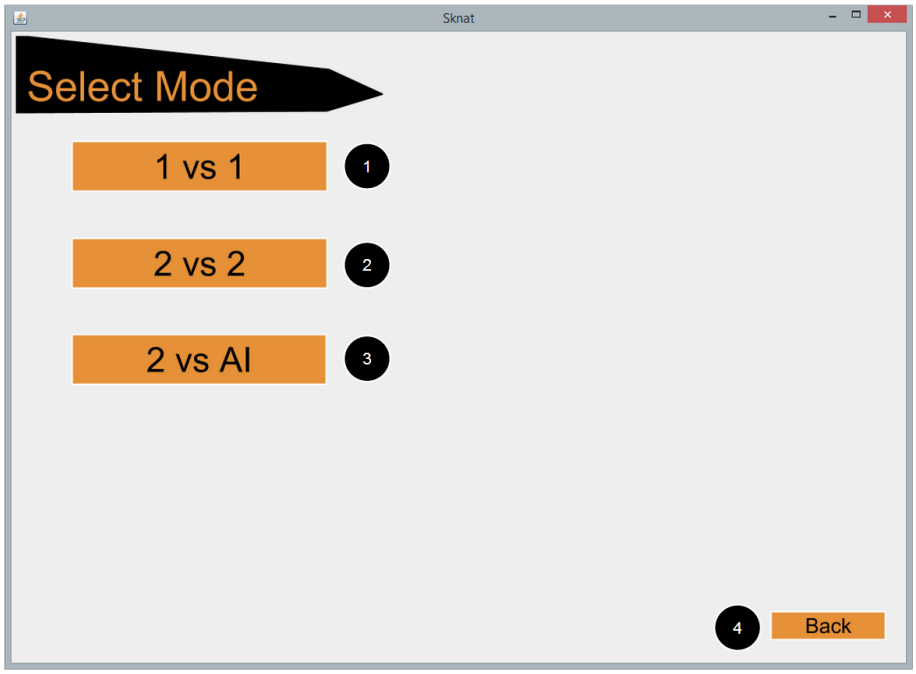


Figure 9: Multiplayer - Game Modes

Object Number	Name	Description
1	1 vs 1	This button will allow the player to create a 1 vs 1 game.
2	2 vs 2	This button will allow the player to create a 2 vs 2 game.
3	2 vs AI	This button will allow the player to create a 2 vs AI game.
4	Back	This button will allow the player to go back to the multiplayer screen.

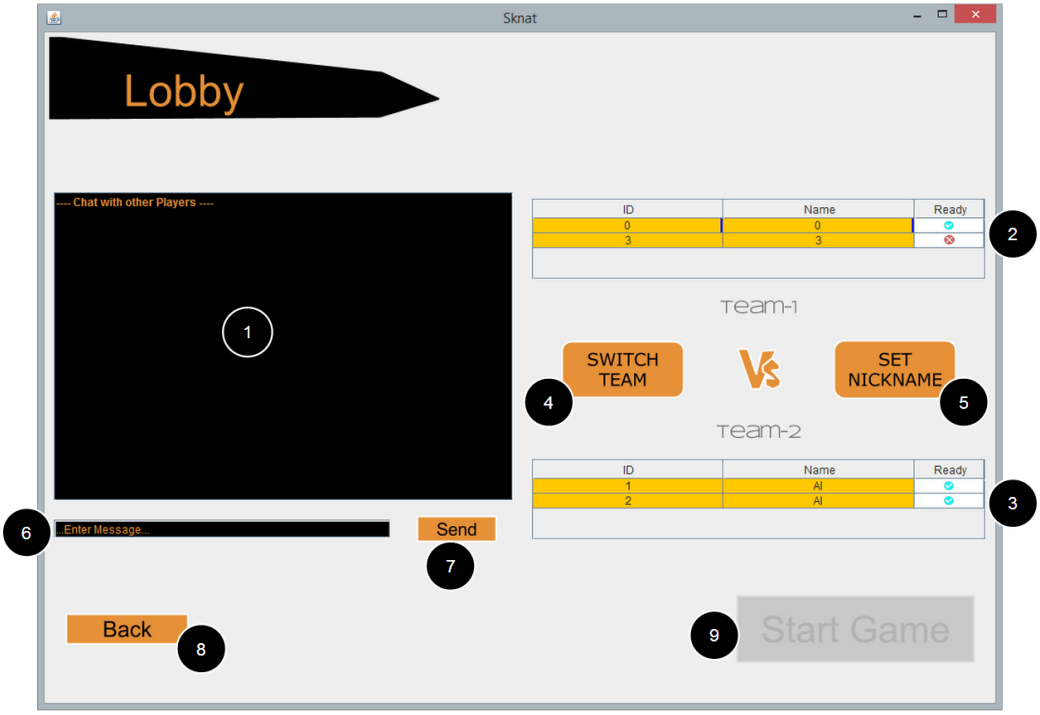


Figure 10: Lobby

Object Number	Name	Description
1	Chat	This area displays all the messages.
2	Team 1	This table shows the players on team one and it also shows who is ready and who is not ready.
3	Team 2	This table has the same functionality as the one for team one but instead it is for team two.
4	Switch Team	This button allows players to switch teams.
5	Set Nickname	This button allows players to set custom nicknames.
6	Message	This is where players have to input their massages in order to chat with other players.
7	Send	This button will send the written message to other players.
8	Back	This button will allow a player to go back to the start menu screen or the multiplayer screen depending on the current game they are on.
9	Start Game/Ready	For the host client this button will start the game and for other client this will be a ready button which allow them to ready up.

Figure 11: Lobby - Set Nickname

Object Number	Name	Description
1	Nickname	This input field will allow the player enter a desired nickname.
2	OK	This button will apply the name change.
3	Cancel	This button will cancel the operation.

4.3. GAMEPLAY

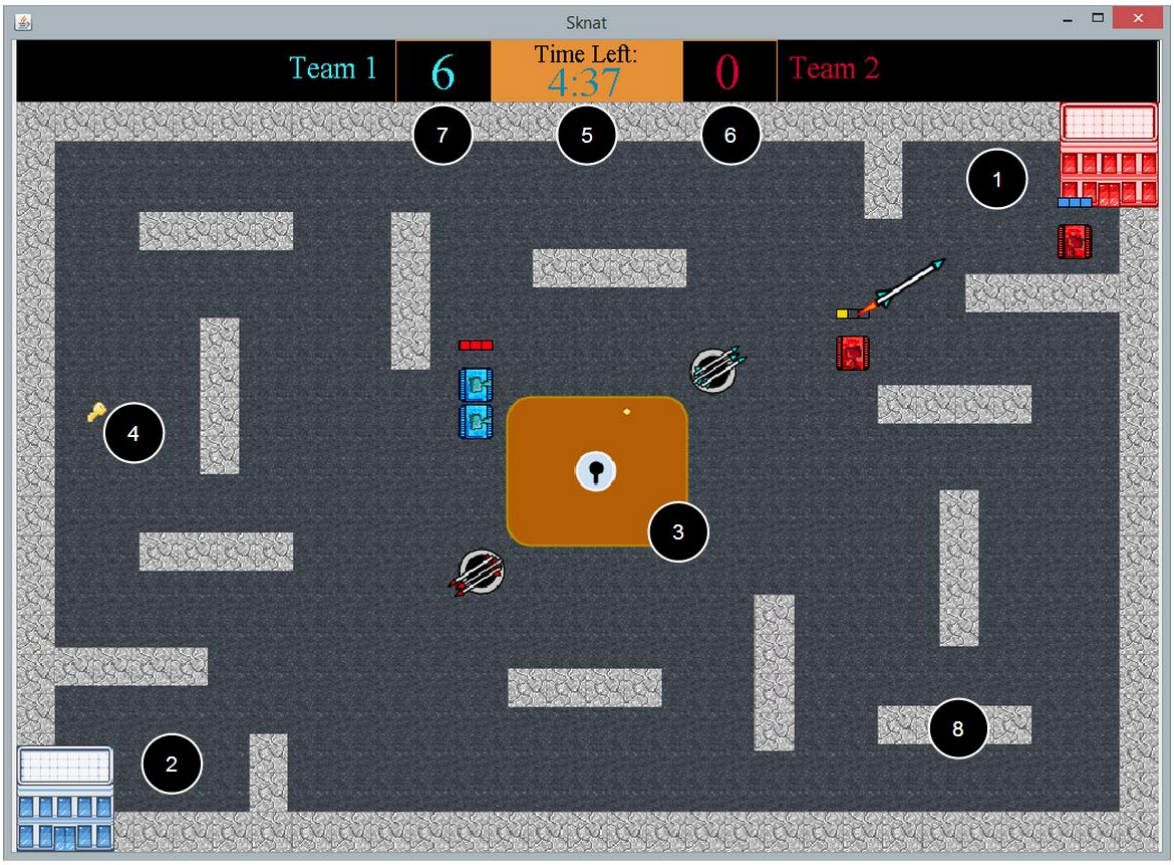


Figure 12: Gameplay

Object Number	Name	Description

1	Team 1 Base	This section shows the base of team 1. This area is used as the point of re spawn for team 1 only.
2	Team 2 Base	Likewise the team 2 base territory functions in the same way as the other team's base territory. The only difference is positioning, team 1's base territory is located in the bottom-left corner of the screen and team 2's base territory is located in the top-left corner of the screen.
3	Objective territory	This territory is part of the main objective of the game. The player has to collect a key which appears in random locations of the map and bring it here in order to gain points for the team.
4	Key	This is the key which appears randomly on the map. As a player you have to take this key to the objective territory in order to get points. If a player carrying the key gets destroyed before reaching the objective territory; he will lose the key and it will appear in another location.
5	Timer	This timer shows the time which is left till the game ends.
6	Team 2 Score	This displays the score of team 2.
7	Team 1 Score	This displays the score of team 1.
8	Obstacles	These are objects which the tanks cannot pass through.

5. NETWORKING

Because we are using a real-time communication and our game is dependent on well-implemented code, we have decided to split the work on it between two team members. Bella and Martin have done pair programming, which allowed them to develop a complete and neat structure of the code. What is more, it made testing and refactoring more quick and optimal.

Throughout the development we strived to follow our initial design so we can achieve good overall architecture of the system. The code is modular and we looked for high abstraction. Thus, it is really easy for the whole team to use and upgrade it.

Our networking is based on communication between a single server and multiple clients. We use the TCP/IP protocol to ensure that all packages have been sent. The decision about which protocol to use TCP/IP or UDP was taken after team discussion and careful testing of both implementations for our game. After performing the tests it seemed that TCP/IP is quick enough to support the information we want to send between the server and clients. However, the way we implemented our code allow us to easily switch between protocols if it was necessary at some stage during development.

On the server we create two different threads for every client - one for sending information to the client and the other one for receiving. After receiving information from the clients, the server does all the computations and updates to the game server and sends all the relevant information about game state changes and events to the clients.

Those updates include:

- Setting the start state of the game
- Updating the movement of the bullets
- Checking if a tank has been hit by a bullet
- Checking if a bullet has reached its maximum range or has hit an obstacle
- Checking if a player has collected the key
- Checking if the player currently holding the key has taken it to the objective
- Computing a new random position for the key after receiving a “goal” point
- Computing and handles all the functionality of the lobby
- Determining the winning teams and the end state of the game

All updates are done twice per frame to ensure smooth game play and pleasant user experience.

We use a concurrent queue in the server to ensure that the messages coming to it from the different players are handled sequentially and none of them is ignored. To ensure that all messages are sent properly we again make use of concurrent queue, but we have one for every player, so the server can differentiate between the different messages that each client should receive. All sending queues are stored in an Array List, so we can access and work with them easily. The use of thread - safe data structures and the concise manner of their implementation made our code much more safe and reliable.

When creating the server object, the number of players is specified. Then a socket is being opened and the server waits for all the players to connect. During this time the players that have already connected can interact with the UI and use its functionality: chat, switch teams, set their ready state, etc. Once everyone is connected, the server waits for everyone to set their ready states and for the host to send an event for starting the game. The start game event triggers all updates listed above.

At the moment, users are required to provide a server's IP in order to enter a game. That is one of our planned extension which we have omitted in order to meet the deadline. We have decided that we want to concentrate on more advanced and major extensions such as AI and tailor them in the best possible way rather than trying to implement randomly smaller ones which do not affect the overall game functionality.

The development of the networking was challenging experience. Nobody amongst our team have been working on implementing a real-time connection. Therefore, Bella and Martin had to do extensive research and lots of tests to ensure that they produce coherent structure of their code. Establishing the connection in time and without any errors was crucial and its completion lead to much more interactive and fast-paced overall gameplay.

6. AI

The game allows two different gameplay modes against AI:

- Single player against another AI player
- Cooperative team of two players, playing against two AI players.

The AI in the game connects to the game server as a normal client. However, instead of waiting for input from keyboard or mouse, the AI simulates movement and actions and sends events to the server the same way as a normal player would generate them.

First we created a very basic AI capable of differentiating between the following states:

- If nobody has the key, try to reach the key.
- If somebody has the key, follow him.
- If the AI is the owner of the key, take it to the objective.

The way the AI is built allows implementing additional states really easy and quickly. Also improving the complexity of the already implemented algorithms is easy with the current architecture. It utilizes all the reusable components of the systems that has been built so far, so it does not add unneeded overhead.

Later Martin and Bella have done further work on implementing the AI. Some of the functionality of our code was also changed in order to integrate a search algorithm.

At first, Breadth First Search was implemented for following the current goal, which varies according to the game state. It allowed the current different functions for following the different goals to be merged into one single function for movement control according to specified position of the goal. The map is being translated into an array of points with X and Y coordinates and it tracks the shortest path to be followed to each goal throughout the game in the same manner for all possible goals.

In order to develop the search we made use of abstraction and modularity as a way to produce a neat and concise implementation which can be easily modified.

Thus, it was easy for us to improve the search to A* in order to speed up the game and to make it more optimal. The shortest path from the current node to the goal is considered as a heuristic. It made the game faster and improved the AI behaviour.

We effectively made use of the current functionality of the game and reused existing behaviours such as movement and shooting.

We developed our AI so it can shoot against players from the opposite team. It differentiate by teams and it shoots only within a specified distance. It checks whether an enemy player is within that distance over a random amount of time.

The behaviour of our AI in terms of movement and shooting is exactly as it is for a real player's tank. It behaves sensibly by trying to gain points and kill opponents. It provides the whole functionality of the game.

7. SOFTWARE ENGINEERING

During the storming stages at our first several meetings we made use of the SCRUM approach to choose idea, distribute tasks and estimate the time needed for each task. Therefore, a decision was never finalized without full agreement of each team member. If there were disagreements we discussed over and over again, outlining pros and cons, reaching for feedback from our tutors or referring to research sources in order to make sure that everyone has agreed that the decision is optimal and worthwhile. Thus, we managed to get each team member even more involved. What is more, by doing so we reassured that each one of us is fully aware of why and how exactly we are going to tackle the different bits of work.

Right after the storming stage our team appointed several meetings for discussions about the architecture and the implementation of our system. We were diligent in specifying and outlining the requirements that we wanted to be covered. The team spent a significant amount of time planning the implementation and deciding on practices and technologies to be used.

Although it felt that we have lost a bit more time than needed only discussing and planning we found that our good initial management was crucial for our later success.

7.1. TECHNICAL PLAN

For the development of our project we have decided to adopt the incremental agile life cycle with elements of extreme programming. We took advantage of the basic principles of this method such as

- Early increments can be planned but the development of later ones can vary in accordance to emerging requirements or approaching deadlines.
- It allows some uncertainty about requirements upon starting the implementation and we were uncertain about some of our requirements initially (e.g. can we allow loss of packages with UDP or can we compromise speed over safety with TCP/IP).
- We wanted to have partial releases frequently (after every planned increment) so we can ensure that the contribution from all team members is compatible.
- We made use of pair programming for some of the complicated tasks (such as networking, AI etc.).
- We did refactoring on a regular basis to ensure that the code is modular, abstract accessible to all team members. It was important for us to save time by producing understandable code which saved us time on our meeting.
- We adopted the principle of collective ownership and continuous integration. So, anyone can make changes and contribute. Working system can be updated daily.
- We specified and followed rigorous coding standards which was very important especially for the implementation of our own parsers (where small error or misunderstanding of functionality leads to serious bugs in the game states).

Furthermore, we used prototyping which helped us to evaluate the feasibility of our system's design. It helped us to identify some issues and to take major decision such as the use of protocols and to clarify some uncertainty around the performance of our game engine.

We applied concurrent engineering to speed up the implementation phase. We developed some of the functionality in parallel. Thus, we achieved our milestones faster and were able to work on some extensions.

7.2. Configuration Management Plan and Risk Management

Prior to the development phase we considered the steps which we might need to take if an unexpected change or event affects our work.

Regarding the change management process we all agreed on:

If the need of a change is identified by any of the team members it will not be implemented prior to approval of the whole team. The change will be proposed and explained on the following team meeting and if approved implementation plan will be specified. The impact of the change will be carefully assessed and all affected artefacts should be carefully investigated.

All team members should understand the change and its effects in depth. The appropriate roles will then be allocated amongst the team to successfully apply the change. The process will be documented and a log of changes will be kept in the common repository.

Back-up copies will be kept for future reference, comparison or as a contingency plan in case of unsuccessful implementation. The project plan and the corresponding documentation will be updated accordingly.

7.2.1. Risks

To handle possible risks we all brainstormed in order to identify the most probable and damaging ones. We analysed them as follows:

Risk	Risk Description	Effect	Strategy	Application of Strategy
Illness	One or more teammates is sick and cannot handle his/her workload.	The project is running late and the work halts.	Impact reduction.	Each bit of work is fully understood by at least two persons, so the workload might be handed over.
Insufficient time for main features	Complications with unfamiliar technologies or complexity of implementation	The project is running late, functionality is omitted, the system is not coherent.	Probability reduction.	Plan carefully, do research and testing prior decisions, discuss with the team
Misunderstanding of lecturers' requirements.	Not meeting the academic staff's expectations.	Lower grade and unsatisfying system.	Probability reduction.	Attend lectures and meetings with our tutor, ask questions and look for feedback.

7.3. Quality Review

Throughout the whole development stage our team was performing continuous quality review. Because of SVN we were always updated with the latest changes each of has done during his or her own working time. So prior every meeting all team members have looked through the latest version and have prepared constructive feedback on the work of the others. By adopting this practice it was really easy for us to maintain high quality of the functionality of the system and clearness of our code. Also, it made easier for us to understand and help each other.

8. TEAM WORK

In the beginning of our project everyone seemed already really excited about the team project and we had many ideas about different games we could develop. During the storming phase we had several meetings in which everyone had the chance to talk and explain his or her ideas, as well as comment on others' ideas. We wanted to have a clear and unified vision of the game, what we want to create and whether we are going to have the time and the ability to create it in the time limit. This later allowed us to progress really quickly with the implementation, as everyone in the team was aware of what the other person was working on and relied on their work.

Deciding on a topic of the game idea took us a long time, because we wanted to choose the most appropriate idea. We took in consideration the complexity of the games, the graphics needed, the networking part, extensibility of the ideas, overall HCI aspects and many others. We realized that this decision is really important about the overall structure of the game and we were really careful about which way to go. After the first meeting we had two different ideas and we asked for an advice from our tutor in order to have more experienced perspective. We wanted to pursue an idea which will showcase our skills and knowledge in the best possible way. After finally agreeing on a topic of the game, each person in our team explained what he/she prefers to work on and we split on different tasks such as GUI, Networking, and Interfaces between them, etc. Every one of us did research on his own and thought of estimates for the tasks. On the next meeting we discussed those estimates and also possible extensions if we finish earlier with the already assigned tasks, and contingency actions if the project runs late.

Having already agreed on the main topic and we split the tasks, we tried to build a first-cut UML class diagram in which we described the basic architecture of the game and how different modules would communicate. We spent a lot of time refining it and trying to make it as modular as possible. This was very hard as we did not have experience creating games and nobody actually knew what were we expected to do. However, spending time on the refining the architecture allowed us to work concurrently on multiple aspects of the game and be faster and more productive as a team. By sticking to our initial design at later stages we were able to produce coherent architecture. Furthermore, this allowed us to have a unified vision of our game throughout the development irrespectively of the different tasks that each of us was working on.

While working on the game we constantly used the SVN. We committed new changes on a daily basis and made sure that every new change is fully functional before it is being uploaded to the main repository. That guaranteed that the latest version on the SVN is always working and can be compiled and built. Uploading code with errors may cause the others of the team to be unable to compile the project and therefore to stop them from working. That is the reason why everyone checked if the project could be compiled and built, before actually uploading it to the SVN. Also we made sure that we write appropriate comments when submitting, which later gave everyone the opportunity to check what has been done since the last time he or she has committed.

Committing changes to repository often and updating to the latest revision always before starting work allowed us to minimise the risk of causing conflicts in the repository. This risk was also reduced by dividing the tasks between the team members in a way that everyone was working on a different part of the game. When this was not possible, we worked together. We found using subversion really useful because we were able to keep each other updated no matter of the time and places different functionality was implemented.

Every week we had at least two meetings that lasted a couple of hours. In the beginning of these meetings each one of the team described and showed what he or she has done during the week. This includes if there was a problem with the game or if we have not defined something very clearly in our first meetings, so that the person developing it does not know what exactly the game is supposed to be. We also shared if anyone has found any bugs in the game, so we can prioritise our work and fix them. At the end of each meeting we checked what everyone was planning on working on next, so that we can avoid conflicts on the SVN or stopping someone from working if he has to wait something else to be done. Apart from our regular two meetings per week, we often met during the weekends if there were any problems or if we needed to work together on a part of the game.

We made use of the TeamWork platform, which allowed us to keep track of the time we are spending on the project. We also had the opportunity to write small notes on what has been done during those hours and keep track of what the others are currently doing. As per requirements each team member tried to work at least 20 hours per week. This seemed a lot to us at the beginning of our project, however we stuck by it and later we realised that if we did not follow the suggestion and have worked less hours, we would have needed to work a lot more during the final weeks to finalise the project or we would not have had the time to add certain extensions that currently make our game more fun and unique.

During our work as a team we did not have any problems with each other. We did even enjoy working together and seeing the game evolve slowly over time. Everyone's opinion was listened to carefully and considered when making a decision about the actual game idea, the design of the game or about an actual way of implementing something. Before making a major decision we made sure that everyone agrees with it. We have learned from each other and improved our programming and personal skills. We found that by sharing knowledge and exchanging ideas we learnt a lot. We also realized the importance of being capable to rely on your teammates in order to deliver a complete big system. The most important thing for us was that we have never been faced with the implementation of such a system before. So we believe that being challenge with a real-world task was really useful and teaching experience. We learned and developed lots of new skills which we will be able to use in our future careers.

9. EVALUATION

Reflecting upon the completion of our project, it is necessary to compare the functionality of our game in it's complete state, to that we initially specified, and from there to assess with hindsight the feasibility of the objectives we set. We also displayed many qualities as a team, but with potential for improvement.

Our team performance has been founded on the back of open discussion and willingness to listen and adopt the ideas of our peers. In the storming phase of our team, we had many concepts, all of which seemed to divide us somewhat equally. As a result, we as a team would regularly shift and discard concepts. We handled this somewhat well given that our outcome emerged as something of a conceptual and technical compromise. I think our early decision making was performed admirably such as to allow our team to perform with momentum and unity when development started (and storming ended). Our storming phase may have been slightly prolonged by a failure to pin a clear definition on our concepts, which allowed the concepts to evolve and merge causing some confusion. Our final concept Sknat was one that we as a group could relate to from existing games that could provide a base concept that we could expand on.

On a week to week basis we referred to our specification. We had pending requirements that needed implementing and would tackle them incrementally in our weekly meetings. Although initially we referenced our Gantt chart for division of labour, it became clear that the requirements evolved organically on a weekly basis. This reflects a weakness in our initial work breakdown structure which was much too specific, and would need updating up to twice a week. Therefore we should have been far more abstract and general in our documented assignment of work.

We worked on all documentation from a central repository and would quality check each other's work before submission. We were responsible in our use of SVN and committed only working code to the trunk.

We achieved the vast majority of our objectives and in some cases more. Our game employs an intuitive interface with an intuitive navigational flow, i.e. from the end state of a multi-player game, a player is returned to the multi-player menu. We were able to achieve many of our non-functional requirements. The game performs well and has the potential to be configured (Unfortunately hard-coded and not available to the user) to a given FPS. In the specification we required that the game be maintainable and extensible. Our modular design was such that it became easy to implement moderate extensions fairly quickly. For example, time-limits on rounds, range limiting on projectiles and a pre-game lobby. We were able to add an auto-player in roughly one week, as the networking allowed the client to be instantiated with AI logic and interact as any other client in the game would.

VoIP chat is one such omission. Although we implemented a text-based chat for the pre-game state, we did not implement VoIP chat, which we regarded as a viable extension. This can be justified through the combination of time constraints and some difficult HCI decisions. We opted to use TCP as our networking protocol as we felt it would provide the best user experience. When compared to UDP, we felt that occasional lag would be more favourable in comparison to the potential loss of synchronisation stemming from packet loss. The reduced speed of TCP was not the limiting factor of our game, as we were able to complete two game state updates per frame. Implementing voice chat over TCP would result in congestion and latency while packets are resent and so would not provide good user interaction. In the specification we decided to omit text-based chat from the mid-game as we felt it would detract from the fast-paced experience of the game.

Our networking generally performs well and achieves our objectives of seemingly ~100% synchronisation of collisions. We were able to tackle the synchronisation issues of UDP by changing protocol to TCP where this no longer became an issue. Our game events are passed as strings with an opcode and parameters. This is useful, particularly when sending values to update the states of pre instantiated objects, i.e. the tanks. Arguably in some situations it would be better to send objects rather than strings, as when updating projectiles for example, we must decode the string containing their new state, remove the previous instance, before re-instantiating based on the values contained in the string. We also became aware after some time of implementation, of external libraries such as GSON which make parsing string representations of objects in formats such as JSON very easy. This would have made for nicer code, and potentially more efficiency. Our parsers however, provide us with flexibility and tailored game events, as well as the ability to send composite messages, reducing the strain on the server. Composite messages allow us to code a tank position and the position of all the projectiles in a single composite message coded as a game state update, something we send at a frequency approaching 100Hz. In terms of security we found a need to balance the issue with HCI. Arguably the most secure implementation would have the server deal with all updates, from collision detection and movement. It was decided however, that the reduced responsiveness of server-side movement was of greater detriment to the game in comparison to the security threat of client tampering. Potentially using a UDP protocol could have overcome the issue of server heavy processing.

Another security issue arose with our method of parsing messages between client and server. When coded, escape characters are used to separate parameters and to separate commands. The issue arose when users are given the freedom to enter a string which will be subsequently coded, i.e. changing nickname or through the lobby chat. This was a particularly interesting bug, as a user could perform an injection attack, and in theory program game events! We overcame this by decoding string based operations differently, but as consequence these types of operations may no longer be included in a composite message. This fix is adequate for our game, but could slightly hamper extensibility where in the future, string based operations may need to be combined.

Despite some omissions of requirements, our final game has achieved the vast majority of our requirements, and our extensions surpass our omissions. This can largely be attributed to our heavily refined class-structure which we worked very hard on in the early phase of development. Our class structure has had surprisingly little change which shows that we achieved clarity and a unified vision of the game before development began. We feel that Sknat is a fun, competitive, interesting and visually pleasing game!

10. SUMMARY

In this report we have tried to document in as much detail as possible our team's development process. We have used modular code and refined class-structures to provide our code with a stable software design which has contributed significantly to our achievements as a team. We designed much of the game's appearance and HCI before development, and section [ref 3.2] is a natural projection from our External Interface requirements (Chapter 3 of our SRS). We have documented our software Engineering practices, in the form of our incremental agile development. We predicted in our specification many of the emerging requirements, such as the lobby state and Auto player. The auto player element was suggested as a desired feature from the client, but due to the nature of the game, we elected to reserve it as an extension. We planned our implementation of this extension, and through a sprint of development, integrated it with the current architecture. We have documented our weaknesses as a team in section [ref 9.0] and identified areas for improvement, from a technical and team performance perspective. I am therefore highly confident, that if our team were to continue to further developments, we would continue to perform at an ever higher standard.

The features that were recommended by our client have been to a large extent covered, and are documented through this report.

- 'The game could/should have features such as:
- HCI issues considered well (e.g., it is easy to get started)
- A team of cooperating players
- A number of competing teams
- Different roles of the different players
- A graphical user interface,
- An auto player (AI component)
- Networking

We have documented our consideration of HCI in all implementation decisions, and have added features based on usability tests (i.e. countdown bars before missile launch). The flow of the GUI to enter the game state are documented and represented graphically [ref 4.1] in such a way that we have determined to be easy/quick to navigate. The game modes we have included supplement our gameplay, reinforcing the feature of co-operation. In the 2v2 mode, upon extraction of the key, it becomes crucial for the other team mate to assist the key holder, protecting them until completion of the objective. We made collaboration more important in the gameplay by increasing the points awarded for completion of this objective, in comparison to operating individually to obtain kills. I.e. objectives are awarded 3 points vs 1 point. I.e. a try vs a drop kick in rugby. We have 2 competing teams at this stage, though the core features of our game present no huge implementation issues with increasing the number of teams who may co-operate. We have a GUI which covers the menu state, pre-game lobby state, mid-game state and end-game state, offering navigation of game modes, lobby communication and a score screen. Our AI is competitive and complex in its implementation and we have carefully considered its difficulty. The game has smooth real-time networking, using our own protocols for connection and parsing of game events [ref 12.3]

11. INDIVIDUAL REFLECTION

11.1. MARTIN MIHOV

Overall, my experience working with the team was enjoyable and challenging. We were enthusiastic and everyone participated equally in the process of creating our game.

From the beginning everyone was enthusiastic about the project and everyone had ideas. Because of this, our first meetings were mainly discussing ideas and brainstorming about what our game should be. Everyone had their chance to speak and describe their thoughts. I had an idea based on a real game, however I had some doubts about it, as it was a bit too complex and I was not sure if it was suitable for the time period that we had. I was given a chance to describe my idea, and I shared both my idea and my doubts. Everyone else was given a chance to talk too and after a few discussions we voted on what do we want to create.

I wanted to work on the networking part of the game, as I am interested in networking and I see it as a challenge. Bella Dunovska also wanted to work on the networking and after sharing this with the others we agreed that it is better two people to work on the networking as it is a big and very important part of our game. Also Bella and I have worked together on previous projects and we were used to working together. After discussing it with the other we were assigned to create the server and the connection with the clients.

We decided to do pair programming as we do not have a lot of experience. This gave us advantage to brainstorm various ways of implementing the server to client connection before actually creating it. We created simple programs to test the speed of the UDP and the TCP connection between multiple clients. The results did not show a big difference, so we did a research about which protocol should we use. UDP was used on games with more than hundreds of players, but using TCP seemed the better choice for a game with maximum of 4 people. After showing the results to the others from the team, we decided that TCP provides enough speed and ensures that packets are being sent correctly. Bella and I started implementing the connection, keeping everything as flexible and modular as possible to allow us to make easy changes in the future if anything arises. We took turns coding and we wrote our code clean and commented.

After establishing the connections, Bella and I started working on the server. The game was supposed to do all computations on the server side. In particular, we were responsible for checking if a tank has collected a key, has been hit or has died, respawning a tank, repositioning the key at a random position and keeping track of the timers for the objective and the game time limit. We started implementing those functionalities and simultaneously working with Samuel Hill on the parsers, which were to process the events information sent between the clients and the server. After implementing the TCP connection we performed various tests to assess its speed and reliability.

When everything was done, we started working on AI for our game. We started building everything modular, so we would be able to perform changes and implement more advanced logic if we had time. We successfully implemented all functionalities for the AI to simulate real gameplay and we started implementing the more advanced search algorithm A*.

If we had more time, we could have developed more intelligent AI and fixed some existing flows in the logic. Also we could have improved the networking communication by optimizing the sending of events with introducing AI logic for players' behaviour.

During our work with Bella, we communicated with the others from our team to discuss problems or ideas that have emerged. To do this we ensured that we have at least two weekly meetings, which everyone from the team attended. On these meetings, we checked the progress that everyone has done and discussed what we were thinking of implementing next. We also had to work together with the others to make custom events being sent from the server to the clients to ensure the correct functionality of the UI.

There were times when not everyone in the team agreed on a given topic, but everyone was given a chance to explain his/hers point of view and to discuss the topic with the others. During this, everyone listened and tried to think what the better way of implementing it was.

11.2. BELLA DUNOVSKA

I strongly believe that working on this project was very beneficial experience for me. To work as a part of a team was the most challenging and improving aspect throughout the work. Our team was enthusiastic and proactive. Each of us has different ideas both about the system's general functionality and the concrete implementation details. That led to extensive communication and sharing of knowledge and ideas. Therefore, I realised how important and beneficial a good team work can be. We attended frequent meetings and we discussed various points of view, also we went through different possibilities about architecture and development, so we all learnt a lot from each other. We incorporated different methodologies and strategy in making decisions and planning, such as SCRUM, evaluation of positive and negative features and brainstorming. What is more, I believe I gained invaluable interpersonal skills and I feel more comfortable in facing future exposures to working as a part of a team.

Once we went through the initial storming stage and we had a clear perspective about our objectives we decided to split the work into logical units. We wanted to develop the project in a neat and concise manner which would allow all team members to collaborate effectively simultaneously. However, we also wanted to stick to our initial architecture as much as possible to avoid lateness and possible errors caused by unforeseen complications.

Therefore, the work was divided and Martin and I decided to work on the networking part. Since it was one of the more crucial yet fairly complicated features of our game we have decided that pair programming would be the safest and most productive approach. Nonetheless, important decisions and implementation details were always discussed with the whole team.

Martin and I managed our time so we could find suitable time slots for both of to work. We concentrated on efficiency, speed of the system and reliability. Thus, we implemented TCP/IP for the networking to ensure that there are no losses when transferring data between different players. The decision to use this protocol over UDP was hard, so we did a lot of research on it. We even implemented both protocols and performed testing in order to decide which one works best for us. We strived to achieve abstraction and modularity so our code is readable and easy to change not only for us but also for the rest of the team. What is more, it gave us the opportunity to easily switch between protocols if needed in future. By pair programming testing and error spotting far more easily.

Once we were ready with the networking we have further improved the functionality of the server. It does all the computations and updates to the game server and sends all the relevant information about game state changes and events to the clients.

In the last three weeks of our development phase Martin and I concentrated on developing an AI. I find that the stage of developing an AI was the most challenging one. However, through collaboration

and diligent work we have managed to implement fully functional AI for all possible game states. Furthermore, it is using A* search in order to be as much optimal as possible. If we had a bit more time I believe that we would have been able to improve it even further and avoid some existing flows.

I am really glad that we are undertaking this module. I believe it gave all of us invaluable experience which we will refer to in our future careers. One of the most improving aspects was that we were faced with the delivery of a real system and the according documentation. What is more, I found that being given the responsibility of managing our time has taught us to some useful practices. Also I enjoyed working on actual complete product.

11.3. Samuel Hill

Working with team A5 has been an interesting experience, culminating in me being able to take significant pride in our final product, from conception to its final form. We have converted accurately an idea into a tangible piece of software, something which I can show to potential employers and can mark as something of a milestone in my career as a computer scientist. The project has given me some insight into the many facets of team work, and in my opinion has provided me with an accurate insight into how small teams in industry would operate. This is a useful experience which I will likely draw upon when entering the real world! I have learnt much from my team-mates over the course of the module. From small coding practices, to major implementation features. For example, before the project I had little concept of thread-safe data structures, and I was phased by the concept of setting up concurrent queues to handle communication between different clients and the server. We also had to discover scale safe implementations for the GUI and so I became familiar with Gridbaglayout, as well as better alternatives to pixel-based absolute rendering.

My skills in teamwork have improved from this module. Particularly my ability to listen to others, consider their ideas as my own and be entirely objective about decision-making. Through the storming phase I feel I learnt to be without ego and to champion the ideas of others. I feel my decision making contributed in the early phases to steer the group in a good direction. We had major implementation discussions about client side, vs server side processing, and ended up with two different client-server models. One in which the server acted almost identically to a client and resembled a p2p network. I expressed concerns over synchronicity and complexity and was able to help the team take the implementation we are using now.

I took particular interest in the development of client side networking and handling server requests. This acted to integrate the fundamentals of the game logic, and the game events which needed to be processed to update local game data, and to update the server's game data from the individual clients. I made the client so that it is instantiated with all the relevant pre-game state data which is sent by the server. This is done by waiting in the constructor for it to be assigned a player ID, which projects across all the client side game logic. The client therefore waits for a single composite string with all the initial data required so that it may instantiate correctly. I created a listener thread which operates on all clients. When it receives a message it is parsed, decoded and executed based on the opcode. I developed a client parser which handles game events which need to be sent to the server. The client contains a game update thread which sends the information regarding tank positions and projectiles to the server twice per frame. I also co-developed the Server parser with Bella and Martin, which I would alter when new game events needed to be sent from the client. As well as the client-side networking I worked on the lobby aspect of the GUI. This required functionality of the lobby meant that game-events had to be introduced, this caused me to need to extend the functionality of the server and to handle these new pre-game events in the parsers. As a player joins a game they are allocated a team ID which is calculated to best balance teams. This is handled by the client and

represented by a table for each team, players may then manually change teams. This is sent to the server, which updates its own revision of the game data and then updates the rest of the clients. Players may also change nickname and update ready status. This added further implementation difficulties, since joining clients had been previously configured to receive just one game event for instantiation with an ID. Now upon joining players had to be made aware of all lobby history before joining. This meant changing significantly the joining protocol of clients.

Overall I have had a positive experience from this module which I owe largely to our team. We all displayed enthusiasm, good work ethic and were supportive of each other throughout.

11.4. TENDAISHE NYAMAPFENI

The time which have spent with team A5 has been one of the most important experiences of my degree. Each week we met up to discuss ideas and critical tasks which needed the attention of the whole team. When the project initially started we spent a lot of time storming ideas and during this time our vision as a team became unified. I now finally understand the importance of writing minutes during meetings because of this.

When we identified what kind of game we wanted to create it was fairly easy to divide the game structure into sub-parts. The major parts of the project were the Networking, Game Engine and the GUI. When we split the tasks I was given the responsibility to work on most of the GUI elements of the game. In order to achieve this I had to become familiar with image editors and in most cases I had to improvise. When I was creating the menus of the game I initially found quite challenging to position elements on the screen because I had to take into consideration window resizing and scaling. However after a lot of struggles with Java swing layout managers I decided to use GridBagLayout for all the menus. This decision proved to a good one as the game started to come together and new features were being put on the table for discussion because changing things was quite easy.

During development I drew a lot of sketch designs for the gameplay and discussed them with the team. I found this helpful because I got feedback from the team which helped me understand where we were going with game and I also believe other members benefitted from this as well.

Overall I think we worked well as a team and feel that the experience which I got from this project is going to be useful in the progression of my career.

11.5. ALEXANDRE PORTUGAL

During the course of this project we developed a game that I can show to my friends and family. A game that I can play in my free time, a game that I will play in my free time. Throughout this module I was able to work in a team with my peers to create something that I am proud to call my own.

In terms of development, I was mostly in charge of creating the game engine and the rendering of the game data, but I also assisted Tendaishe in some of the User Interface. Technically, I did not find any of the parts assigned to me to be hard, but I believe I have learned a lot and grown as a programmer during this experience.

During this project, I was able to use my interest and knowledge in math and use it in our project. This can be seen in several classes, for example, in the method that calculates the projectile motion and in the method that calculates the angle that the tanks cannon should be facing. The latter while may seem simple, is complicated by the scaling and translating of the game screen when it adapts to different screen sizes.

I also learned a lot about graphics as I was responsible for how the game was drawn on the screen. At the start of the project we had to decide whether we were going to use OpenGL or not, which led me to do a significant amount of research on it as I had no previous knowledge on how to do it. This culminated in the team deciding not to use OpenGL and using Graphics2D instead.

As a team, I was lucky to be placed in such a friendly group, and I had a lot of fun working with them. Had I had a different group, I am positive I would not have enjoyed the module as much as I did. The team atmosphere was always extremely positive, even amongst moment of pure despair, mostly due to the school's WIFI not working.

If I had to do this again, I would. This has been, without any doubt, the most fun module I have done. From our meetings to the last "sprints" before major deadlines (like the prototype) our team remained enthusiast and happy. I could not have hoped for a better team. My only complaint about this module is the use of the teamwork website, which I believe to have an extremely poor user interface.

12. APPENDICES

12.1. APPENDIX 1 – UML DIAGRAM

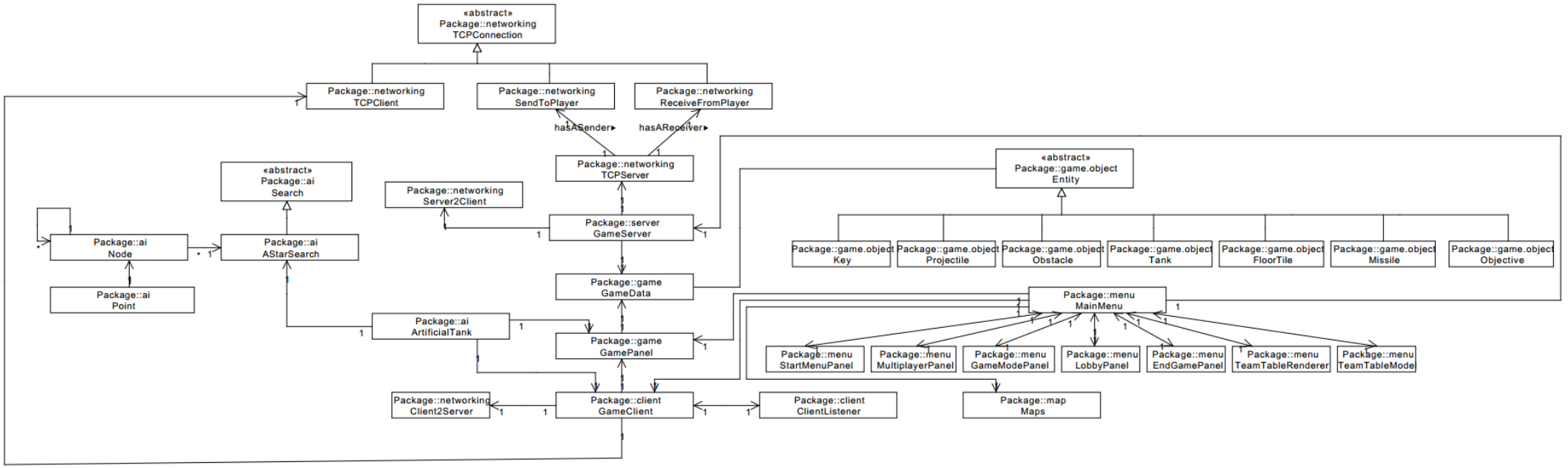


Figure 13: UML Diagram

Figure 13 shows the UML diagram that highlights the class structure. A standalone copy of this diagram which is much easier to will be uploaded to canvas.

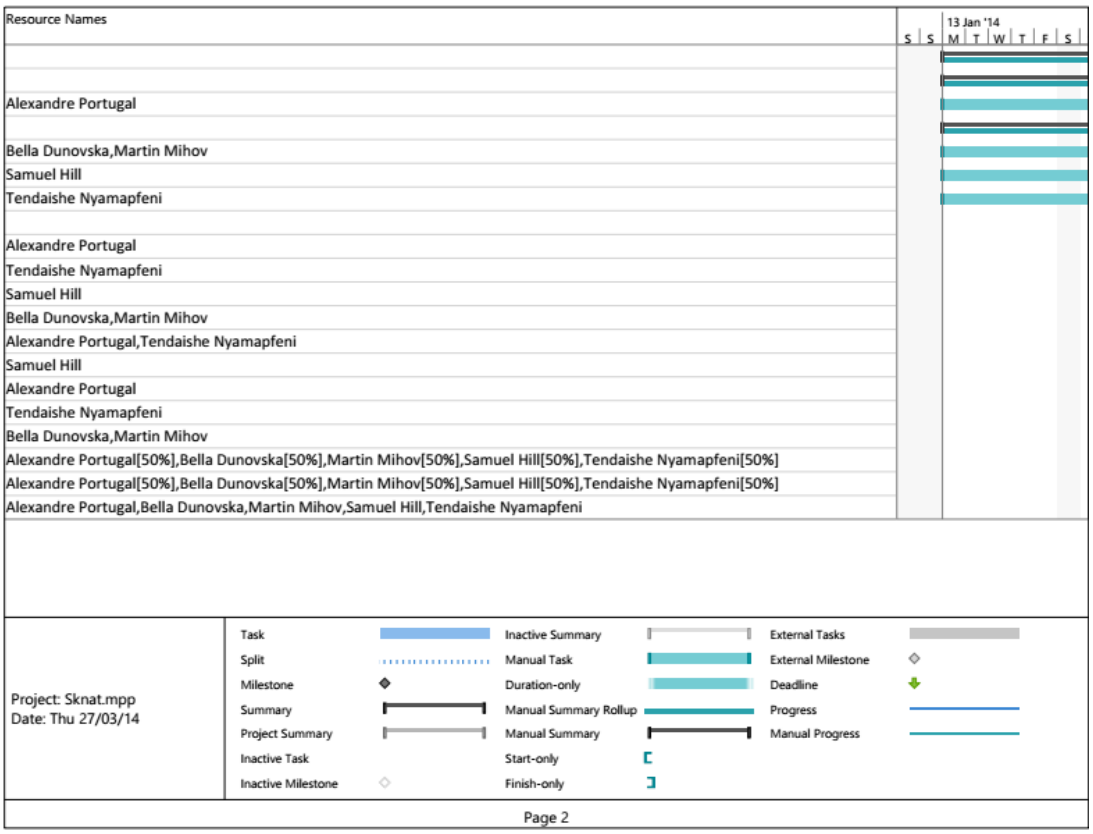
12.2. APPENDIX 2 – GANTT CHART

ID	Task Mode	Task Name	Duration	Start	Finish	Predecessors
1		**Sknat**	**55 days**	**Mon 13/01/14**	**Fri 28/03/14**	
2		**Functional prototype**	**25 days**	**Mon 13/01/14**	**Fri 14/02/14**	
3		Game Engine	25 days	Mon 13/01/14	Fri 14/02/14	
4		**Networking**	**25 days**	**Mon 13/01/14**	**Fri 14/02/14**	
5		Server developmer	25 days	Mon 13/01/14	Fri 14/02/14	
6		Client connection	25 days	Mon 13/01/14	Fri 14/02/14	
7		Menu Interface	25 days	Mon 13/01/14	Fri 14/02/14	
8		**Extensions**	**21 days**	**Sat 15/02/14**	**Fri 14/03/14**	**2**
9		Creating a map	5 days	Sat 15/02/14	Thu 20/02/14	3
10		Improve the menu int	5 days	Sat 15/02/14	Thu 20/02/14	7
11		Improving the pasers	11 days	Sat 15/02/14	Fri 28/02/14	
12		Server extensions	11 days	Sat 15/02/14	Fri 28/02/14	4
13		Chat interface and bu	9 days	Fri 21/02/14	Wed 05/03/14	
14		Functionality and inte	11 days	Sat 01/03/14	Fri 14/03/14	
15		Implement Sounds	7 days	Thu 06/03/14	Fri 14/03/14	
16		Create a new game lo	7 days	Thu 06/03/14	Fri 14/03/14	
17		AI	11 days	Sat 01/03/14	Fri 14/03/14	
18		Final testing	6 days	Sat 15/03/14	Fri 21/03/14	8
19		Presentation preparatio	6 days	Sat 15/03/14	Fri 21/03/14	8
20		Writing final report	6 days	Sat 22/03/14	Fri 28/03/14	19

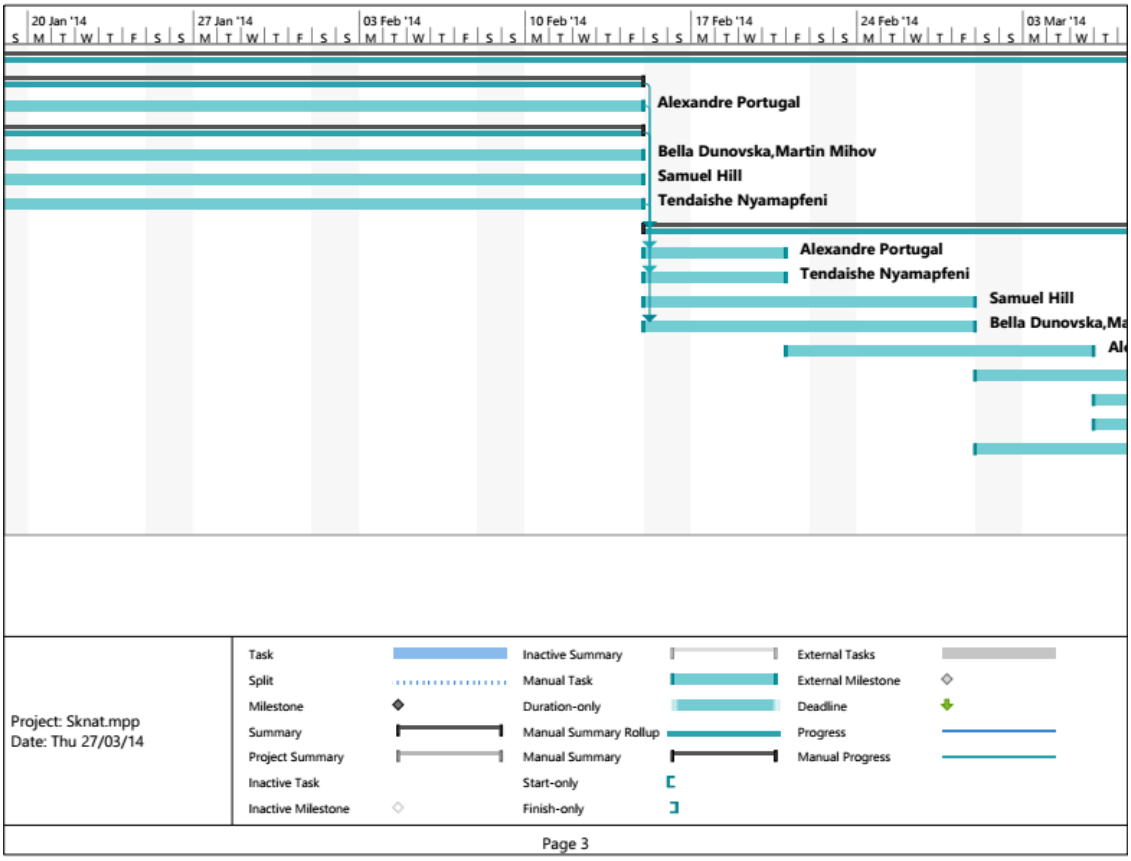
Project: Sknat.mpp
Date: Thu 27/03/14

Task, Split, Milestone, Summary, Project Summary, Inactive Task, Inactive Milestone, Inactive Summary, Manual Task, Duration-only, Manual Summary Rollup, Manual Summary, Start-only, Finish-only, External Tasks, External Milestone, Deadline, Progress, Manual Progress

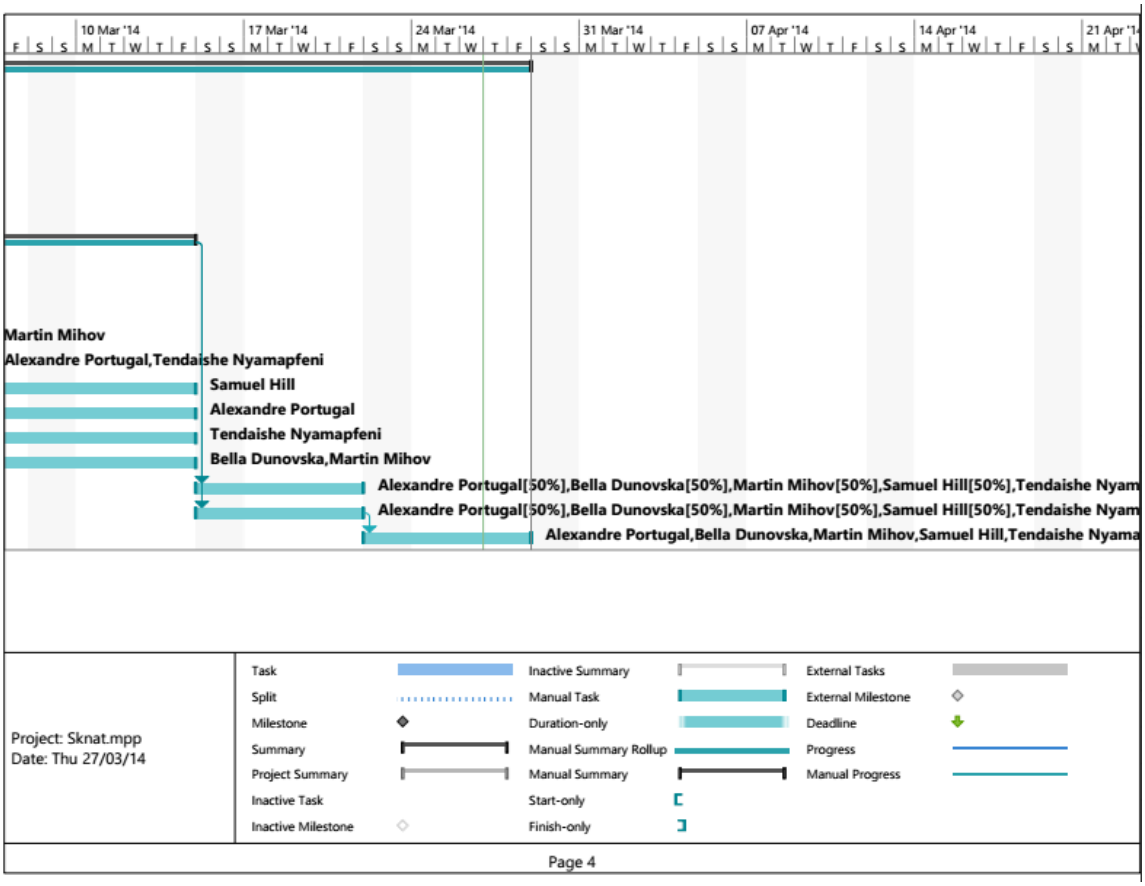
Page 1



Page 2


20 Jan '14
27 Jan '14
03 Feb '14
10 Feb '14
17 Feb '14
24 Feb '14
03 Mar '14
Alexandre Portugal
Bella Dunovska,Martin Mihov
Samuel Hill
Tendaishe Nyamapfeni
Alexandre Portugal
Tendaishe Nyamapfeni
Samuel Hill
Bella Dunovska,Ma
Project: Sknat.mpp
Date: Thu 27/03/14
Task
Split
Milestone
Summary
Project Summary
Inactive Task
Inactive Milestone
Inactive Summary
Manual Task
Duration-only
Manual Summary Rollup
Manual Summary
Start-only
Finish-only
External Tasks
External Milestone
Deadline
Progress
Manual Progress

Page 3


10 Mar '14
17 Mar '14
24 Mar '14
31 Mar '14
07 Apr '14
14 Apr '14
Martin Mihov
Alexandre Portugal,Tendaishe Nyamapfeni
Samuel Hill
Alexandre Portugal
Tendaishe Nyamapfeni
Bella Dunovska,Martin Mihov
Alexandre Portugal[50%],Bella Dunovska[50%],Martin Mihov[50%],Samuel Hill[50%],Tendaishe Nyam
Alexandre Portugal[50%],Bella Dunovska[50%],Martin Mihov[50%],Samuel Hill[50%],Tendaishe Nyam
Alexandre Portugal,Bella Dunovska,Martin Mihov,Samuel Hill,Tendaishe Nyama
Project: Sknat.mpp
Date: Thu 27/03/14
Task
Split
Milestone
Summary
Project Summary
Inactive Task
Inactive Milestone
Inactive Summary
Manual Task
Duration-only
Manual Summary Rollup
Manual Summary
Start-only
Finish-only
External Tasks
External Milestone
Deadline
Progress
Manual Progress

Page 4

12.3. Appendix 3 - Network Sequence Diagram