

Group Project NO1

Project Plan

Author: Mark Peters, Jim Finnis
Config Ref: SE_N06_MAN_01
Date: 2011-10-11
Version: 1.0
Status: Final

CONTENTS

CONTENTS	2
1. INTRODUCTION	4
1.1 Purpose of this document	4
1.2 Scope.....	4
1.3 Objectives.....	4
2. OVERVIEW OF PROPOSED SYSTEM	5
2.1 Client.....	5
2.1.1 Friends.....	5
2.1.2 Map Viewer.....	5
2.1.3 Login/Register.....	6
2.1.4 Model.....	6
2.1.4.1 Friends.....	6
2.1.4.2 Map	6
2.1.4.3 Session.....	6
2.1.5 Database Connection.....	6
2.2 Protocol	6
2.3 Server	6
2.3.1 Request handler.....	6
2.3.2 Model.....	6
2.3.2.1 Map.....	6
2.3.2.2 Session manager.....	6
2.3.2.3 Friends.....	6
2.3.2.4 Clue generator.....	7
2.3.3 Update handler.....	7
3. USE-CASES	8
3.1 Client use-cases	8
3.2 Unauthenticated user use-cases	8
3.2.1 Log in.....	8
3.2.2 Register.....	9
3.3 Authenticated user use-cases	9
3.3.1 View friends.....	9
3.3.2 View notifications.....	9
3.3.3 Respond to friend request.....	9
3.3.4 Issue friend request.....	9
3.3.5 View rich list.....	9
3.3.6 View status.....	10
3.3.7 View map.....	10
3.3.8 Leave clue.....	10
3.3.9 Dig	10
3.3.10 Bury.....	10
3.3.11 View clue.....	10
3.4 Administration use-cases	10
3.4.1 Delete user.....	10
3.4.2 Delete treasure.....	10
3.5 Server use-cases	11
3.5.1 Log in.....	11
3.5.2 Register.....	11
3.5.3 Get friend list.....	12
3.5.4 Get map data.....	12
3.5.5 Periodic update check.....	12
3.5.6 Bury treasure and place clue.....	12
3.5.7 Clue generation.....	12
3.5.8 Dig check.....	12
3.5.9 Friend request.....	12
3.5.10 Accept friend request.....	12
3.5.11 Disconnect.....	12
3.5.12 Robbery.....	12
4. USER INTERFACE DESIGN	13

4.1	User navigation	13
4.2	GUI design	13
4.2.1	Log In screen.....	13
4.2.2	Register.....	14
4.2.3	Friend list and notifications screen.....	14
4.2.4	View map.....	15
4.2.5	Help screen and extra screens.....	15
4.2.6	Menu and status bar.....	16
5.	GANTT CHART	17
6.	RISK ANALYSIS.....	18
	DOCUMENT HISTORY	19

1. INTRODUCTION

1.1 Purpose of this Document

The purpose of this document is to describe how the group project will be completed, by translating the client's requirements into a set of objectives.

1.2 Scope

This document layouts the project schedule, major tasks, milestones and resources of the group project.

1.3 Objectives

1. To describe the architecture of the proposed system.
2. To describe interactions between users and the software.
3. To describe the appearance of the software and how the user interacts with it.
4. To specify the main tasks of the project, responsibilities of group members for completing them and the schedule on which they will be completed.
5. To describe possibly problematic parts of the plan and how such problems could be diminished.

2. OVERVIEW OF PROPOSED SYSTEM

Figure 1 shows the diagram of the high level architecture for the system. The server side will be using a PostgreSQL database and PHP and the client side will be programmed using Java.

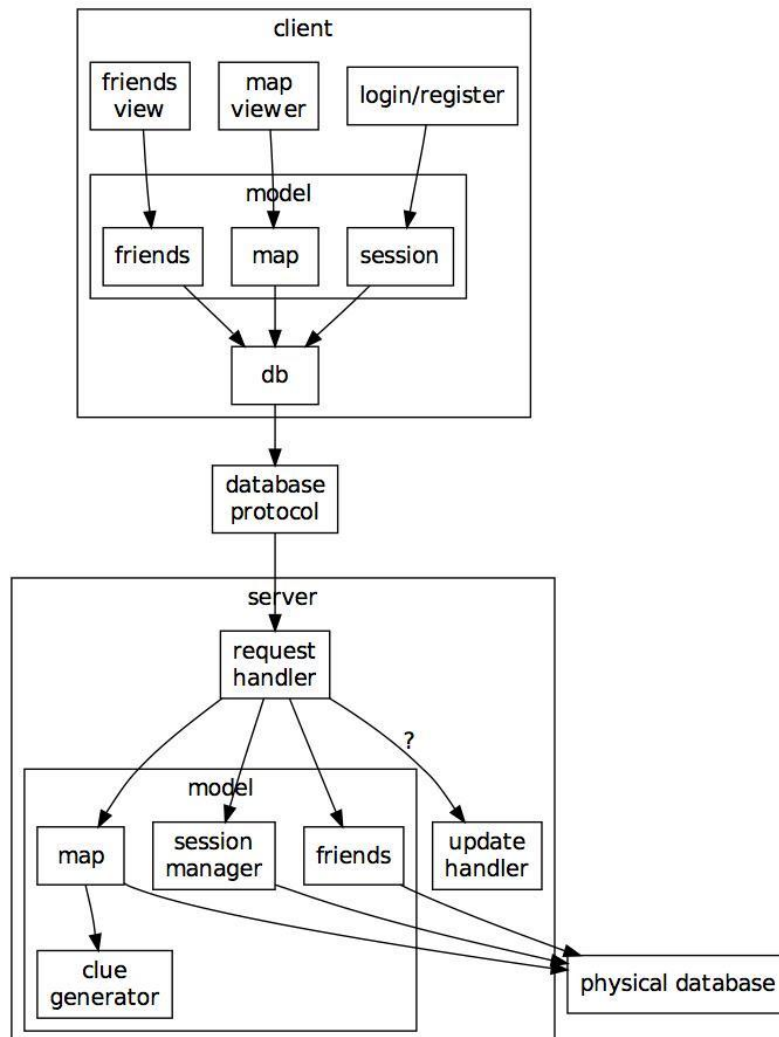


Figure 1: High level architecture

2.1 Client

This section describes the architecture of the Android application. This is going to be made up of several screens, as described in the user interface design.

2.1.1 Friends

This is one of the two primary screen elements, displaying the friends list and notifications. It will probably consist of a single view or activity reading data from the friends part of the model.

2.1.2 Map viewer

This is the other primary screen element, consisting of the map display. It will read the landmark/clue information from the map model and display each landmark's position relative to the user on a zoomable canvas.

2.1.3 Login / Register

This is really two screens but they're both very simple compared to the others and so can be considered to be a single architectural element. They consist of a few fields and buttons, whose actions connect to the session model and verify an existing user or create a new one.

2.1.4 Model

These modules deal with the data the client application holds, keeping that data in synchrony with the data kept on the server, via server requests. The idea is that the screen modules mentioned above do not interact directly with the database but instead work through these modules, which can be unit-tested.

2.1.4.1 Friends

This section of the code maintains the list of friends on the client side, keeping it synchronised with the database via the database interface.

2.1.4.2 Map

This section of the code maintains the map data (probably as a list of landmarks), keeping it synchronised with the database via the database interface.

2.1.4.3 Session

This section of code keeps information about the logged-in user (including whether a user is logged in or not). It will also keep some kind of session ID which needs to be passed to the server with each request once the user is logged in.

2.1.5 Database connection

This part of the code handles the database connection with the server.

2.2 Protocol

This is a definition of the protocol passed between the client and the server, consisting of definitions of the requests from the client and their responses from the server.

2.3 Server

The server responds to requests (probably HTTP) from the client, and responds to the client in a similar manner.

2.3.1 Request handler

This unpacks the HTTP request and passes it to the appropriate part of the server, which will pack up a response and return it to this module in some form and pass it back to the client.

2.3.2 Model

These modules handle interfacing with the database to retrieve and update data, and also clue generation when treasure is buried.

2.3.2.1 Map

This part of the server code handles retrieving data from and updating data within the map database.

2.3.2.2 Session manager

This code handles a list of logged-users, ensuring that each client's request is matched with a user. PHP has a built-in module for session handling.

2.3.2.3 Friends

This handles interfacing with the database to retrieve and update friend information.

2.3.2.4 Clue generator

This generates clues when a “place treasure and clue” request is received, producing an appropriate complex number expression which is then converted into pirate code. It is called from the map module.

2.3.3 Update handler

This code is possibly one of the trickier parts; it involves sending information about changes to the database (friend requests, new clues etc.) to other users who are also connected to the database at the same time. This will involve catching changes to the database and pushing them to other active sessions.

3. USE-CASES

3.1 Client use-cases

Figure 2 shows the use case diagram for the system client. It describes both normal user use cases, and those which are available only to the administrator. The administrator cases may be handled by a separate command-line system.

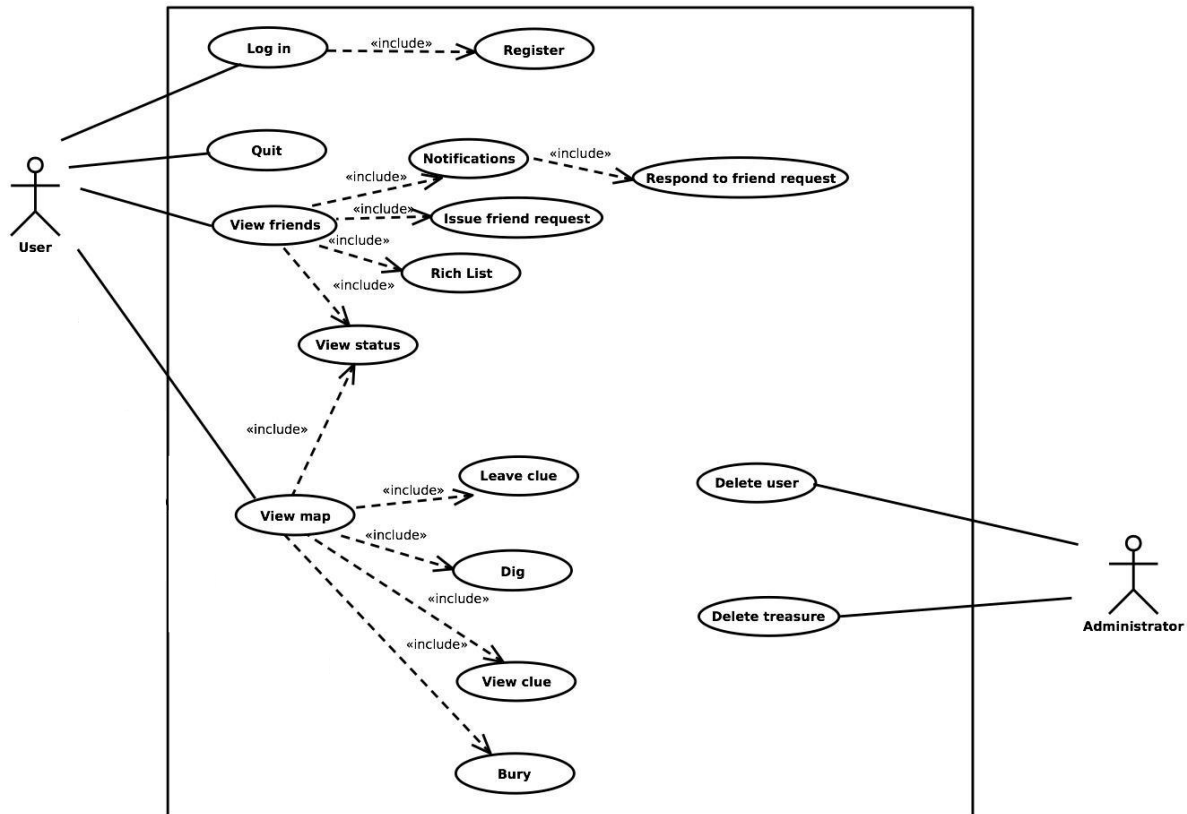


Figure 2: Client use-cases

The use cases divide into two types: those which apply to any user, and those which apply to administrators only. In addition, only the Log in and Register use cases can be performed by unauthenticated users (users who have not logged in).

3.2 Unauthenticated user use-cases

3.2.1 Log in

This is a simple user/password authentication. The user enters a username and password, a login request is sent to the server, and an authentication token is returned if the login was successful (or an error code of some kind if not). Once the authentication token has been returned, the user can use the authenticated cases. Users should log in using an email address.

If the request is not successful, a dialog box should appear and the log in process should start again. In addition to the user name, password fields and the "log in" button, there should be a "create new user" button which will go to the Register use case.

A note on security models, it is generally considered a very bad idea to send passwords over the network without any encryption. It is proposed that the following security model is used:

1. User will enter username and password.
2. Client application will encrypt the password, and send the user name and encrypted password to the server as part of a "login" request.
3. Server receives the request and,

- (a) Retrieves the user record for that user. If no such user exists, it will send back a failure message and abort.
- (b) Compare the encrypted password in the request to the encrypted password for that user.
- (c) If the two match, the password must be that with which the two registered with. If they don't, send back a failure message and abort.
- (d) Create a session (PHP has code for doing this) and send the session ID back to the client.
- (e) The client will then use the session ID (or token) in all further communication. I realise that this is slightly outside the scope of a use-case document, but it's a useful thing to pin down early, and this is a useful place to do it.

3.2.2 Register

The user is requested to enter a new email address, their full name, and two passwords, the second password being to confirm correct password entry. When the user presses "OK" a request is sent to the server. If the server responds with "user exists", an error dialog is shown and the register process exits, returning to the "Log in" case. If the user record was created successfully on the server, the user should automatically be logged in and will enter the "View friends" screen.

3.3 Authenticated user use cases

None of these use cases should be available to unauthenticated users. Most of them involve sending requests to the server, which should be accompanied by session IDs or an equivalent form of authentication token.

3.3.1 View friends

This is one of the "main game views", showing all the friend related tasks together. It is a scrollable list of all the players with whom the user has a friend relationship.

3.3.2 View notifications

This shares the same screen as "View friends". It is a list of recent events triggered by other friends. These include:

- 1) Treasure found by a friend, with location and amount.
- 2) Friend requests made to this user.

The list of notifications should be sorted by date, scrollable, and with friend request notifications given priority. It would also be nice if the following notifications could be made:

- 1) Treasure buried by a friend.
- 2) Friend request confirmed by recipient.

3.3.3 Respond to friend request

This is done by tapping on a friend request in the notification list and clicking "Yes" on a confirmation "Are you sure?" dialog.

3.3.4 Issue friend request

This is done by pressing the '+ new friend' button on the friend view which will open a dialog into which the user can enter an email address, along with OK and Cancel buttons.

On clicking OK, the name will be sent to the server, which will verify that the user exists. If not, a failure message is returned and the server aborts. If the user exists, that user will be sent a friend notification from this user.

3.3.5 View rich list

This function is effectively subsumed into the main friend list, as friends will be sorted on the friend list by wealth order.

3.3.6 View status

This is a bar at the bottom of both the friend and map views. It shows the current number of buried treasures, how much cash is held and how much cash is buried.

3.3.7 View map

This is the other “main game” view. It shows the virtual landscape corresponding to the real landscape around the user. The map can be zoomed and panned using the standard Android methods (pinch, “unpinch” and swipe.)

The user's location is shown as a dot. Icons depict location of landmarks, which hold clues for the user to read. The map will be updated from the server whenever a server-side change occurs.

3.3.8 Leave clue

This button is enabled on the map screen when a burial has been done. It allows a clue to be posted for the most recent burial. The user is required to select an amount to bury. Provided the amount is valid, a server request is sent containing the amount, with the burial and clue locations. The user can still move the map while the request is pending (which is indicated by an icon.) The result of the transaction with the server is shown in a status message.

3.3.9 Dig

This button is disabled if a dig is in progress. The button allows the user to try to dig. The server is contacted and any nearby treasure returned - if the dig was successful, the treasure is deleted, notifications are sent, and the user's hoard is incremented. A time delay is added either before or after the server is contacted. The user can abandon the dig at any point. (FR9) (FR4)

3.3.10 Bury

This button is disabled if the user's hoard is zero or a recent burial hasn't had a clue created yet. It allows the user to bury treasure at their current location, popping up a dialog box requesting the amount, with OK and Cancel buttons. Nothing is sent to the database until the corresponding clue is added. If the amount is invalid, the dialog will not accept it. (FR9)

3.3.11 View clue

When a user is close enough to a landmark, a green ring appears around that landmark. Double-tapping on the landmark reveals the clue hidden there. This opens a dialog showing the clue in pirate code. (FR8)

3.4 Administrator use-cases

Figure 1 also shows the use case diagram for the system administrator. The Administrator user is an optional extra. These are possible functions required to administer and test the system, and could be implemented by writing a command-line tool to manipulate the database directly. There may be additional functions.

3.4.1 Delete user

This allows users and all their associated data to be deleted.

3.4.2 Delete treasure

This allows misplaced treasure to be deleted.

3.5 Server use-cases

Figure 3 below describes the interactions the client program has with the server. It does not describe the administrative functions, since these will probably be performed by a special, direct client. Note: Although there is a little figure of a person on the diagram, this is actually two programs talking to each other.



Figure 3: Server use-cases

3.5.1 Log in

The server attempts to authenticate a user with the credentials passed in with the request. If successful, it creates a session for that user, using a session management mechanism in PHP.

3.5.2 Register

The server attempts to create a new user, assuming one does not already exist with the given name (which is an error). If successful, it also logs the user in.

3.5.3 Get friend list

This is done automatically on login. The server sends a list of the user's friends and their hoard values.

3.5.4 Get map data

This is done automatically on login. The server sends a list of all landmarks (clues).

3.5.5 Periodic update check

This means that any changes are sent to actively connected clients as soon as they happen. This could be done by having connected clients poll the server occasionally.

The data sent back to the client will typically be friend requests, "other notifications" and map changes.

3.5.6 Bury treasure and place clue

A new clue is generated and the treasure/clue pair added to the database. This involves clue generation, a complex task.

3.5.7 Clue generation

A new clue is generated as part of adding some treasure to the system. The clue is generated from the clue location, the treasure location, and the value.

3.5.8 Dig check

The database is queried to see if some treasure is at the given location. If so, an acknowledgement is sent to the client with the treasure's value, the client's hoard is increased by the value, and the treasure and the clue for that treasure are removed from the database.

3.5.9 Friend request

A friend request is added to the system. Periodic update check should ensure that the notification is sent to the recipient immediately, or very quickly, if they are online. Otherwise it is sent when they next connect.

3.5.10 Accept friend request

The server creates a new friend relationship and involves the sending of a notification.

3.5.11 Disconnect

The user is disconnected, and any session or periodic update check data is removed. This may also be done automatically, not just when a user quits.

3.5.12 Robbery

Although not included in the diagram, because it's entirely an internal function, the server should occasionally rob users of a random proportion of their horde. When this happens, a notification should be sent to the user.

4. USER INTERFACE DESIGN

4.1 User navigation

Figure 4 below is a flow diagram which describes how the screens of the program are navigated. The number of help screens can be increased if needed and these extra help screens would also be navigated by a next arrow icon.

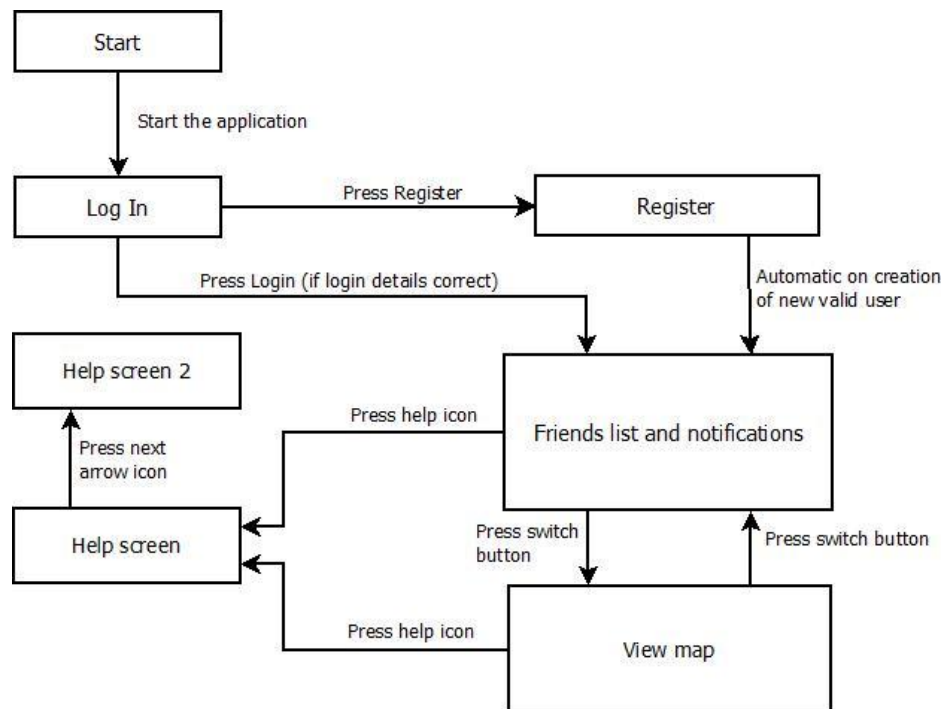


Figure 4: Flow diagram of the program screens.

4.2 GUI design

The diagrams contained in section 4.2 are the designs of the Graphical User Interface. They are simple visual representations of the screens of the Android application. This is to give an idea of what the screens will look like but are not a final design.

4.2.1 Log In screen

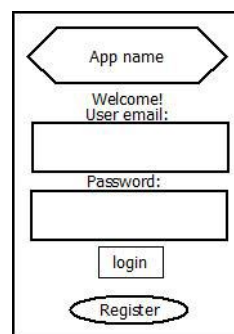
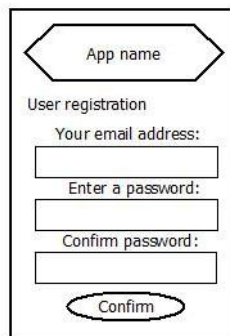


Figure 5: The Log In screen

Figure 5, above, is the first screen that welcomes the user when they start the app. It consists of the title of the game, fields for existing users to enter their email and password details to login, and buttons to log in or register as a new user.

4.2.2 Register

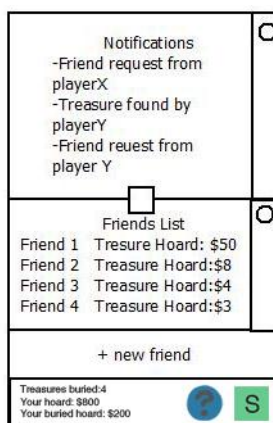


The diagram shows a vertical rectangular screen. At the top is a hexagonal input field labeled 'App name'. Below it is the text 'User registration'. Then, there are three stacked input fields: 'Your email address:', 'Enter a password:', and 'Confirm password:'. At the bottom is an oval button labeled 'Confirm'.

Figure 6: User registration screen

Figure 6, above, describes the user registration screen. It consists of the title of the games and fields to enter the user's email address, full name and two passwords to ensure correct data entry. There is a "confirm" button to enter these details. If the details are valid a new user is created and they automatically enter the friends list and notifications screen.

4.2.3 Friend list and notifications screen



The diagram shows a vertical rectangular screen. The top half is a scrollable list titled 'Notifications' with three items: '-Friend request from playerX', '-Treasure found by playerY', and '-Friend request from player Y'. Below this is a small square button. The bottom half is a scrollable list titled 'Friends List' with four items: 'Friend 1 Treasure Hoard: \$50', 'Friend 2 Treasure Hoard:\$8', 'Friend 3 Treasure Hoard:\$4', and 'Friend 4 Treasure Hoard:\$3'. At the bottom of the screen is a bar with the text '+ new friend' and a status bar at the very bottom showing 'Treasures buried:4', 'Your hoard: \$800', and 'Your buried hoard: \$200' next to a globe icon and a green square icon with the letter 'S'.

Figure 7: Friend list and notification screen

Figure 7, above, is one of the main game screens. At the bottom is the menu and status bar, see section 4.2.6 for more information. The rest of the screen is split horizontally into two scrollable lists, the list of notifications in the top half and the list of friends in the bottom half.

The notifications are presented in order of the most recent. See section 3.3.2 for the types of notifications there are. When a notification is a friend request, it can be responded to by pressing it which will open a dialogue box asking them to confirm the friend as such.

The friends list contains a list of the friends of the user in order of treasure hoard so it can also be used as the scores table to compare which players are richest. At the bottom of the friends list and above the menu and status bar is a large "+ new friend" button, which when pressed opens a dialogue box to add a new friend by inputting a friend's email address.

A possible optional extra is that by pressing and dragging the square in-between the friends list and notifications list the user can make one list take up more of the screen than the other to suit their needs at the time.

4.2.4 View map

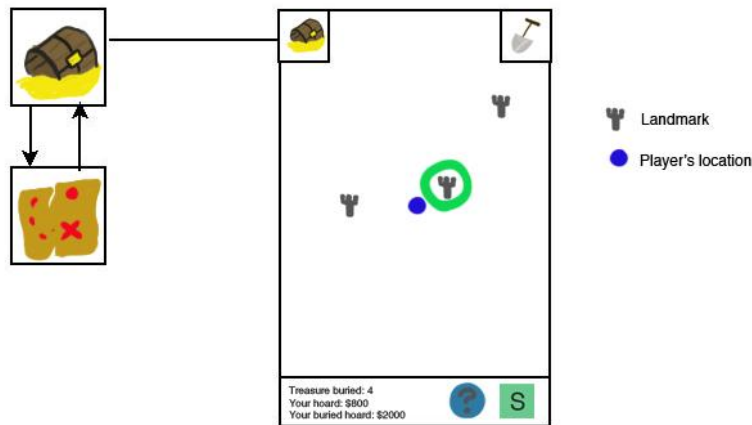


Figure 8: Map screen

Figure 8, above, is one of the main game screens. Originally it is a blank screen except a dot in the centre indicating the players GPS coordinates in respect to the rest of the map.

If the user's friends have placed clues then landmark icons will appear on the map should the user be close enough. These landmarks are the locations of the clues, when the user is close enough to one of these landmarks the icon of the landmark will have a green ring appear around it and the user will be able to read the clue. With this information the player can try to find the treasure by pressing the "dig" button in the upper right corner of the screen. If the player is close enough to the treasure their hoard will be incremented with the amount of treasure they collected.

The user can move to a location in the real world and bury treasure by pressing the "bury" button in the top left corner of the screen. On pressing this button, a dialogue asks a user how much they wish to bury and will take that amount from their hoard if valid. The "bury" button toggles to a "leave clue" button, as indicated to the user by the icon changing from a buried treasure chest to a treasure map. The user cannot bury again until they leave a clue. On pressing the clue button, the user will leave an icon of a landmark at that location, which will contain the clue to the treasure they just buried. The "leave clue" button will toggle back to a "bury" button, and they may bury more treasure again.

4.2.5 Help screen and extra help screens

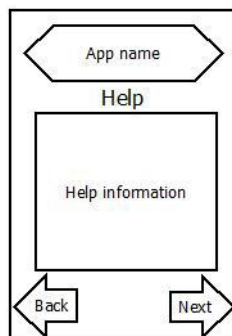


Figure 9: Help screen

This screen consists of the title of the game, information on how to play the game and arrow icons to navigate to the next help screen and back. Pressing the back arrow icon on the first screen takes the user back to the friends list and notifications screen. Pressing the next arrow leads to another help screen with different information.

4.2.6 Menu and status bar



Figure 10: The menu and status bar

This is not a screen but an interface used on both main game screens. It displays the number of treasures that have been buried, the amount of treasure the player has on them (their hoard) and the amount of treasure the player has buried. It also has two buttons, the button with a question mark on it will send the player to the help screen, while the button with the “S” on it, switches the player between the map view and notifications screens.

5. GANTT CHART

Figure 11 below is a Gantt chart, a visual representation of the project schedule. It describes the project milestones, team responsibilities and task dependencies.

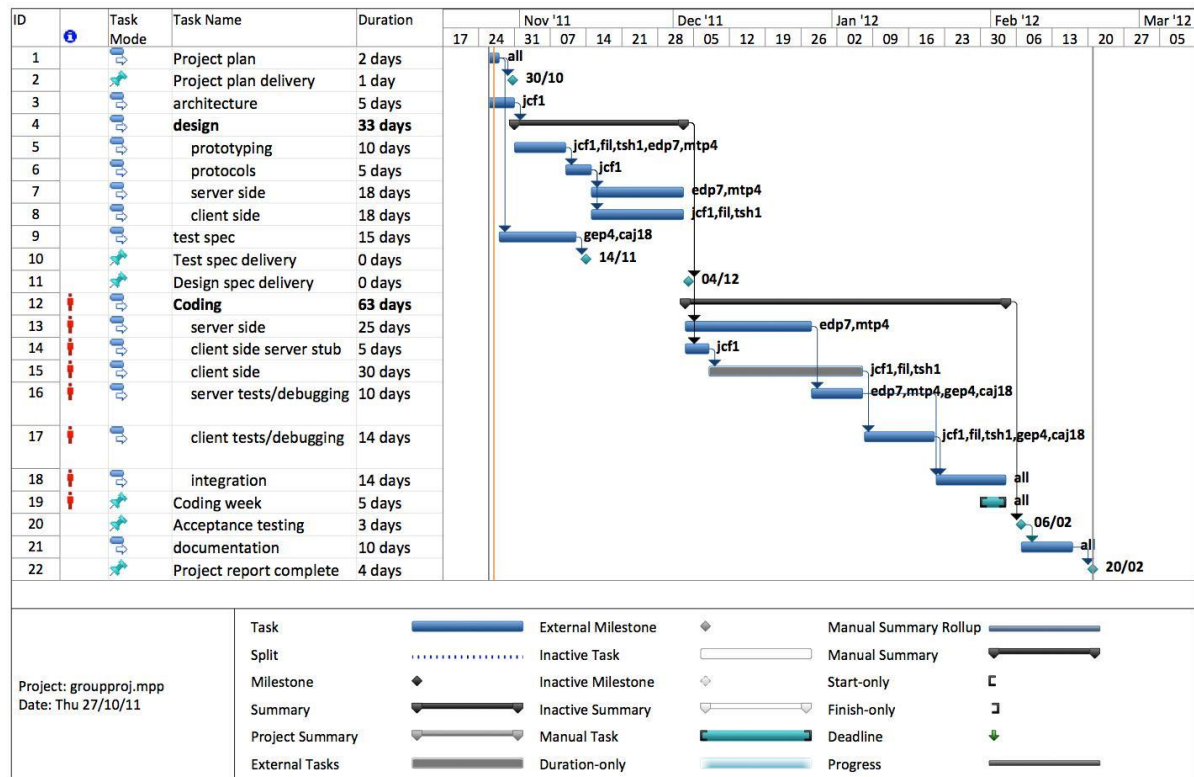


Figure 11: Gantt chart of the project schedule.

6. RISK ANALYSIS

Expected difficulties of the project and how to mitigate them are reflected in the following table, figure 12.

Task	Hazard	Risk level	Protective measure/ Alternative procedure
Creating an educational game for children (10 year olds).	They may find the game itself too difficult or they are playing incorrectly.	Low/Moderate	Making information on the help screen clear and easy to understand. Making sure clues are not too hard. Dialogue boxes to inform users of errors to help them realise they are playing incorrectly or system errors.
General programming of both client side and server side	Programmers may be unable to meet deadlines, due to running short on time due to assignments, illness and certain tasks taking longer than expected	Moderate	The product schedule is generous on the programming tasks, and team responsibilities have been delegated carefully.
The update handler	One of the harder parts to code. Changes to the database need to be sent to other users who are also connected to database at the same time.	Moderate/High	Will involve strong communication between client side and server side teams.
Clue generation	Algorithm needed to generate pirate clue from the clue location, the treasure location and the value of treasure buried.	Moderate/High	Client team need to be aware of this tasks difficulty, and be ready to devote a large amount of coding time towards it.

Figure 12: Gantt chart of the project schedule.

DOCUMENT HISTORY

<i>Version</i>	<i>CCF No.</i>	<i>Date</i>	<i>Changes made to document</i>	<i>Changed by</i>
1.0	N/A	30/10/11	N/A - original draft	MTP4