Norwegian University of Science and Technology

IT2901 - Project 2

Don't Panic!

A Serious Electronic Board Game

Authors:
Jens Even Blomsøy, Stian Aurheim, Jørgen Foss Eri
Sindre Svendsrud, Adrian Arne Skogvold, Jim Frode Hoff

Customer:
Ines Di Loreto

Supervisor:
Mohsen Anvaari

April 10, 2013

Contents

1 Introduction

1.1 The course

The main goals in this course are to experience and learn how to work on a development project as a team. In addition, the team has to answer to a customer, as software development companies often do, which stands out from other projects in the past. This is an advanced course and it is expected that knowledge obtained from previous courses is used, especially the development courses such as Informatics Project I and the collaborative System Development project.

The group has an appointed guidance counselor as well as a customer. The counselor will be available for answering questions regarding the project management in general, and push the group to reflect on its decisions and review the work done. Status reports will be delivered regularly, so the counselor can stay up-to-date with the work in the group.

During the course, several project reports are scheduled for delivery; the preliminary project report, the mid-term project report and the final project report. Working on and delivering these reports will help in the planning and development of the project, and feedback will be given from the counselor. The grading of the project will take the final project report into consideration, as well as the final product.

1.2 The team

The team consists of six students of Informatics at NTNU:
Stian Aurheim

- Third year bachelor in Informatics
- Main experience in Java. Some experience in Python, PHP, HTML, JavaScript

Jens Even Berg Blomsøy

- Third year bachelor in Informatics
- Programming languages worked with: Java, Python.
- Main knowledge in System Development, system architecture and system documentation.

Jørgen Foss Eri

- Third year bachelor in Informatics
- Experience with Java, Python, JavaScript/HTML5/CSS3 and general web development

Jim Frode Hoff

- Third year bachelor in Informatics
- Programming languages worked with: Java, Python, PHP, JavaScript and general web development

Adrian Arne Skogvold

- Third year bachelor in Informatics
- Programming languages worked with: Java, C#, Oz, Actionscript

Sindre Svendsrud

- Third year bachelor in Informatics
- Experience with Java, Python and C++

1.3 Problem description

The customer has developed a paper prototype [Figure 1] of a board game called Don't Panic. The game is designed to help crisis workers make the right decisions during a crisis in a city. It is a turn based, collaborative multiplayer game where the players take actions to stop the inhabitants from panicking. After the game is finished, an expert will review the actions with the players to evaluate whether their choices were sound.
Our customer wants an electronic version of the board game. The electronic board game should maintain the social aspect (both physical and verbal) of a regular board game. In addition, a replay function will be added, to make it easy for the expert to review the game with the different players. The physical version of the board game takes a lot of work setting up and maintaining, as it is time consuming to move the pieces and update the panic levels. The electronic version will automate all of this.

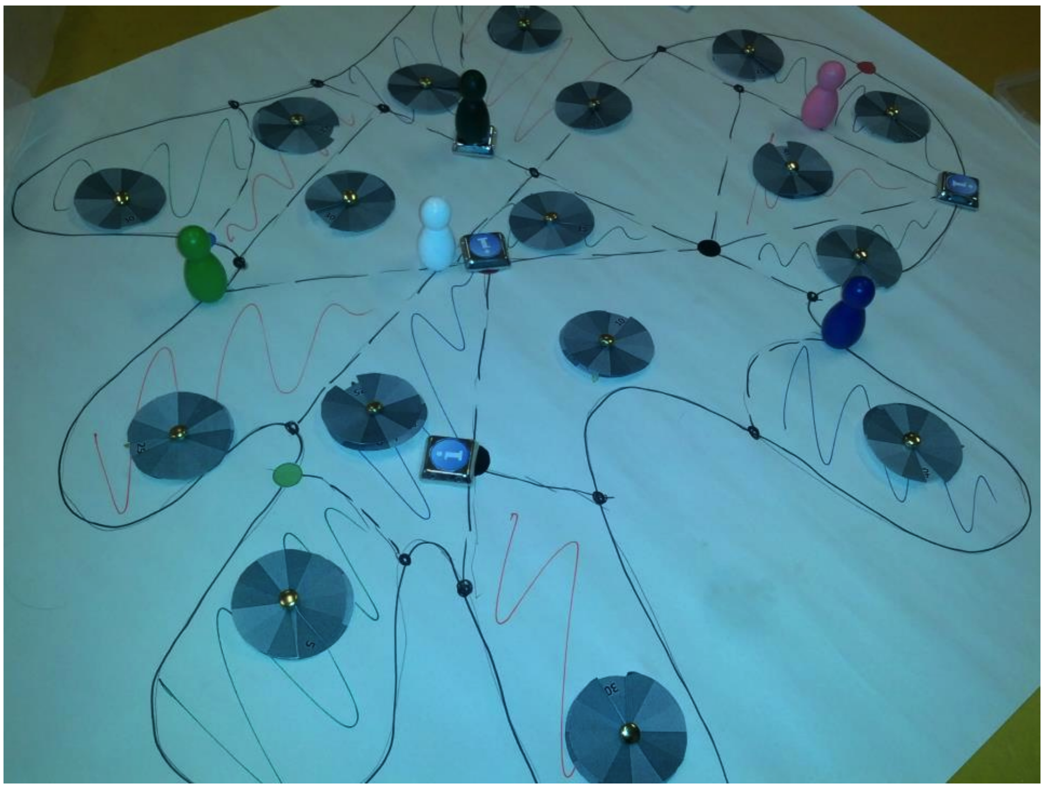

Figure 1: Picture of the paper prototype

1.4 Constraints

Only a few of us have some past experience with HTML5 and JavaScript
The game is to be completed within one semester (21 January - 27 May)

1.5 Customer and supervisor

The customer for this project is Ines Di Loreto, a researcher in the Department of Computer & Information Science at NTNU. The supervisor assigned for this project is Mohsen Anvaari, a PhD candidate in the same department.

2 Requirements

2.1 Functional requirements

- The game interface should provide experts with the ability to efficiently manage and control the players.
- The players must be able to interact with the client using an I/O device.
- The game interface should have elements for managing game sessions.
- All game objects that exist longer than a single task, must be created and kept in storage. An object is anything from the players' data to an invisible trigger object.
- Game rules should be accepted as parameters in input. These parameters should be set by the experts through the management client.
- The expert should be able to follow a game session and comment on the tracking of the game, but should not be able to participate.
- The expert should have the ability to effectively monitor the server.

2.1.1 Use cases

The use cases are mainly based on the functional requirements of the game and are a graphical representation of the users' interactions with the board game. They document all the different ways in which the user can interact with the game.
A detailed set of use case diagrams and textual use cases are provided below.

ID	**01**
Name	Login Player
Goal	To be connected to the server
Actors	Player, server
Start requirements	None
End requirements	- The player gets logged in. - The game is displayed.
Case	- The player gets prompted with the login window. - The players gives login-info. - The player is now logged in.
Alternative Case	Wrong password
Previous Use Case	None
Spawned Use Case	05

Table 1: Use Case: Login player

ID	**02**
Name	Login Expert
Goal	To be connected to the server
Actors	Expert, server
Start requirements	None
End requirements	- The expert gets logged in. - The expert view is displayed.
Case	- The expert gets prompted with the login window. - The expert gives login-info. - The expert is now logged in.
Alternative Case	Wrong password
Previous Use Case	None
Spawned Use Case	03

Table 2: Use Case: Login expert

ID	**03**
Name	Game Setup
Goal	To create a successful game session
Actors	Expert, server
Start requirements	The expert is logged in
End requirements	The expert is able to create a game setup The expert is able to save the game setup
Case	The expert creates the appropriate map for the game. The expert adds the wanted board pieces. The expert manages the zones, the number of people and panic levels. The expert manages the cards, adds the wanted cards to the game. The expert adds the wanted number of players to the game. The expert assigns roles to each player. The expert sets the starting point for each player.
Alternative Case	None
Previous Use Case	02
Spawned Use Case	04, 05

Table 3: Use Case: Game Setup

ID	**04**
Name	Watcher
Goal	To get a non player version of the game
Actors	Expert, server
Start requirements	The expert is logged in, a game is running
End requirements	The expert is able to watch the wanted game
Case	The expert selects the game to watch from a list of games The server provides a game window in which the expert is not participating as a player
Alternative Case	None
Previous Use Case	02
Spawned Use Case	None

Table 4: Use Case: Watcher

ID	**05**
Name	Join Game
Goal	To successfully join a starting game
Actors	User, game session, server
Start requirements	A game has been created by the exper
End requirements	A user is able to join the appropriate game
Case	The user clicks on join options for the game The game asks for user info The user joins the game
Alternative Case	The user gives incorrect info and is not added to the game
Previous Use Case	03
Spawned Use Case	06, 07

Table 5: Use Case: Join Game

ID	**06**
Name	Move game pieces
Goal	To move a game piece to a wanted location
Actors	Player, game board, game session, server
Start requirements	The player has joined a game The player in question has the turn
End requirements	The player is able to move the selected piece to the wanted position.
Case	The player uses the game pad to select the wanted object. The player drags the object to the wanted location. The game board is updated.
Alternative Case	The player selects an immovable object The player moves the object to an unobtainable location
Previous Use Case	06
Spawned Use Case	None

Table 6: Use Case: Move game pieces

ID	**07**
Name	Use information cards
Goal	To use an information card to affect the board
Actors	Player, Game Board, Game Session, Server
Start requirements	The expert has created a game The player is logged in The player is part of a game The player has an information card
End requirements	The card effect is carried out on the board The player does not have the used information card
Case	The player selects the information card The information card effect is carried out on the board The player loses his information card
Alternative Case	None
Previous Use Case	05
Spawned Use Case	None

Table 7: Use Case: Use information cards

ID	**08**
Name	Use player action
Goal	The player uses an action and the game board is updated
Actors	Player, Game Board, Game Session, Server
Start requirements	The user is logged in The user is a player in the game A game is in action
End requirements	The player uses an action The effect is updated on the board
Case	The player selects an action with the gamepad The player selects a zone or a node The action is used on the target The game board is updated
Alternative Case	None
Previous Use Case	05
Spawned Use Case	None

Table 8: Use Case: Login expert

2.2 Non functional requirements

All non functional requirements comply with the definitions as stated in the ISO 25010 standard (replacing ISO 9126). Only relevant requirements are mentioned in this report.

2.2.1 Quality in use

2.2.2 Efficiency

Like regular board games, actions should not be difficult to execute. The players are working against the clock (the panic increase timer). Hence, when designing the user interface, one of the aims should be to minimize the number of clicks required.

2.2.3 Context coverage

The system should be flexible enough to accommodate individual experts' preferences and needs in their simulations. By relying on the settings given by the expert through the expert interface form, the best possible flexibility can be ensured.

2.2.4 Product quality

1: Functional suitability
Functional completeness should be achieved to include the core functionality of the board game, as well as the functionality specific to the electronic version, like the expert interface and panic- and people management.
Core functions must be without game-breaking bugs to ensure functional correctness.

2: Operability
Ease of use is considered important, as the users should spend time playing the game and learn how to manage panic, rather than how to operate the game.
By exploiting recognisability from classic board games, a lot of interaction can be made intuitive, given that most people already know how to play board games.
Users of the game will most likely not be as proficient with computers as "gamers" in general. Therefore, it would be a good idea to make the game accessible without having to install anything other than an internet browser.

3: Transferability
The client should be usable on as many platforms as possible (Mac, Windows, Linux, Mobile platforms), and in the best possible case be able to interact with devices such as Arduino. HTML5 with node.js was chosen for this reason, as it can run on nearly any device without the need for time consuming installation procedures.

2.2.5 Technical requirements

These requirements have been copied from the "Don't Panic" specifications provided by the customer.

Don't Panic DPS has to meet to the following requirements:
- All interaction between the server and client SHOULD be performed using well documented protocols and standard protocols.
- The DPS Game rules SHOULD be platform independent. Consequently, high level languages such as Java, Processing, Python COULD be considered as good candidates.
- The overall architecture SHOULD be scalable to run multiple Game sessions in parallel without decreasing the quality of already running games sessions.

-Already existing frameworks for game development for such as Unity, Microsoft XNA Game Studio, or management tools such as RedMine COULD be used as platforms to help speeding up the development of the game. The choice should be driven by a framework comparison analysis considering both technical requirements and already existing skills/experience among group participants.

3 Prestudy

* *denotes the chosen alternative, if there were multiple choices.*

3.1 Environment

The first and probably most important decision to be made in this project was to decide on a language or software to serve as a development environment. There were several good alternatives, but finding one which was suited to both the task and accessible in terms of experience needed from the developers became a minor challenge.

3.1.1 Front end language

Java
The first and most obvious choice for any project on NTNU is Java, as it is the language used in most courses and all the team members are familiar with it. However, there are several issues when choosing Java for the front end. First of all, the ugly nature of the Swing framework is well known (although there are some alternatives). In addition, an applet in Java requires a plugin to work, if it is to be deployed on the web. A desktop version was an option, but that would limit the game to desktops only. Personal preference was also an issue with some of the developers, mostly regarding the Java's static typing and rather verbose syntax.

Unity
A program like Unity would lessen the amount of code on the client side, as creating a user interface and board for the game would be almost drag-and-drop, with some scripting to handle mouse interaction and data transfer with the server. However, in a boardgame that does not require more than 2D graphics, Unity would be overkill. It also requires a plug-in to work.

XNA
This is a game development framework in C# that runs on the xbox360 and windows platforms. Given that none of the team members were familiar with C#, and the restrictiveness in terms of portability, this was not a good option.

HTML5 *
The choice eventually fell on HTML5 as the best front end technology. With the proper browser, it runs on nearly all platforms, which was a key requirement for the game. Its simplicity in drawing 2D objects with the canvas

element would be useful and make development faster. A couple of the team members were already familiar with HTML5 and JavaScript, which would present and manipulate the canvas. Furthermore, it would be useful for the whole team to increase the knowledge of JavaScript, as it is the most widespread programming language on the web.

3.1.2 Back end language

Java
Again Java presents itself as the most obvious option, as its use as a back-end language with the Spring framework is widespread. Personal preference was one of the decisive factors, as well as productivity. Compared to the alternatives, it seemed like this would be the most time-consuming option.

Python
With web frameworks such as Django, Flask and others, Python was quickly named as a decent option. Most of the team members had some experience with the language and frameworks, and writing Python is quick (and fun, according to some). As an engine for a real time application though, it was considered unsuitable.

Node.js / JavaScript *
This platform was the most foreign to the team, yet it showed a lot of promise. First of all, the entire project could be written in one language, namely JavaScript, which would really hammer the concepts of prototyping (in JS) and give the team more experience with this popular language. Node is event-driven and relies heavily on asynchronous functions, which suited well with the imagination of producing an event-driven game. In the end, these factors made this option the best fit for making "Don't Panic".

3.1.3 Data transfer protocol

JSON *
Given that all the writing would be done in JavaScript, choosing JSON (JavaScript Object Notation) was easy. As JSON looks exactly like JavaScript objects, it was easier to understand and work with JSON than for example XML, which looks more like HTML and did not really suit the needs for a simple protocol to send commands and JS objects through.

3.1.4 Database

MongoDB
An analysis was made of MongoDB which actually stores the data in JSON documents instead of tables. This would be convenient, since it had already been decided to use JSON for data interchange. However none of the team members had any knowledge of MongoDB and it would be time consuming to learn it.

NoSQL
NoSQL is efficient for storing a large amount of data that does not necessarily need to be structured. It does not offer any functionality beyond storage (like keys). It is faster than relational databases like MySQL. However there was no need for storing a large amount of data, and the data was structured. Therefore this was not a very good option.

MySQL*
MySQL is the world's most popular and used open source database. It is used by e.g. facebook, wikipedia and google. MySQL is a relational database management system and therefore fits well with the data. The team members had some experience with MySQL from earlier courses such as TDT4145 Data Modeling, Database and Database Management Systems and IT1901 Project 1. In addition the team knew that IDI could provide a MySQL database on their server, which was convenient. MongoDB seemed like a promising alternative, but this option was considered more time consuming than MySQL. That is why MySQL was chosen.

3.2 Frameworks

Socket.io *
This is the go-to JavaScript library for real-time web applications using Websockets. It contains a client-side library that runs in the browser, and a server-side library for node.js. Like node.js, it is event-driven.

Node-mysql *
This is a node driver for mysql. It enables connection to mysql database with JavaScript.

jQuery *
The jQuery library simplifies access to the DOM, provides animations and

easy element content manipulation.

Express *
This is a web development framework for node.js, that simplifies access to routing, requests and sessions.

3.3 Versioning

Subversion
This is often the standard versioning system used at NTNU, as a repository is provided by IDI, and the team members have used it in several courses already. It is a centralised system and more mature in its development than Git, the alternative. However, it is slow in comparison. Branching is cumbersome and if the central server is not available, it can cause significant trouble.

Git *
In Git, all clones of the repository act as a back up, and the system itself is distributed, where a clone on Github (in this case) acts as a communication channel between the users. Some of the team members already had experience using Git, and found it a lot easier and faster to setup and use in practice.

3.4 Project management tools and processes

Google Drive
Google Drive is a file storage and synchronization service provided by Google that enables collaborative editing of the project documentation. For this project documentation Google docs was used. As this is a collaboration based tool, it suited the structure of the work method.

Dia
In addition to the documentation tools for diagrams provided by Google Drive a program for creating various diagrams, Dia was used. This program has templates of almost all UML designs.

Wbstool
Wbstool was used to make the work breakdown structure chart.

Kanban
Kanban is a method for developing software with an emphasis on just-in-time delivery, while making certain not to overload the developers of the system

with work. At the heart of Kanban lies the Kanban board; a visual process management tool consisting of a Kanban board and cards. Each card represents a task that can be assigned to members of the development team. The board is divided into sections, separating tasks that have only been defined, from tasks that are in progress and tasks that are finished. Using this system, any member of the group can create and assign tasks to other group members, and keep an eye out for who is doing what at any given time.

3.5 Existing solutions

Since this is an original board game developed by the customer, there are no alternative electronic solutions of this game already developed. However, there is a large number of other board games that have been adapted into a digital version using various technologies.

Examples:
- Chess: http://plainchess.timwoelfle.de/
PlainChess is a chess implementation built completely using HTML5 technologies. The game engine is written in JavaScript and relies on the frameworks jQuery and jQuery UI, and games can be played both with and without the use of an internet connection.

- Planet Sudoku: http://planetsudoku.com/
Planet Sudoku is a robust, customizable HTML5 Sudoku game supporting different kinds of Sudoku rules and difficulty settings.

- Bombermine: http://bombermine.com
Bombermine is a massively multiplayer online adaption of the classic strategic puzzle game Bomberman by Hudson Soft/Konami. The game was made with HTML5 and JavaScript, using the AngularJS and async.js frameworks. Bombermine won the Best Web-Only Game at the Mozilla Game On 2013 competition, and although it is not as similar to a traditional board game like Don't Panic is, it really shows the possibilities for HTML5/JavaScript games.

After having reviewed these example games, it was clear that the chosen technology of HTML5 and JavaScript was a good choice for creating the game.

List of Figures

List of Tables