Revenge of the Moon - The Gathering Storm

Subject 2 - Minebombers clone

Authors:

- 84745F Miro Nurmela miro.nurmela@aalto.fi
- 290658 Henri Niva henri.niva@aalto.fi
- 84312L Joonas Lipping joonas.lipping@aalto.fi
- 78740E Roope Savolainen roope.savolainen@aalto.fi

Last update:

2013-31-10

Specification of requirements

All minimum requirements will be implemented. The game will feature single player and hotseat multiplayer, fog of war on the map, a store, collectible treasures, several game rounds, a few different terrain types (such as tar, ice, and walls of varying strength), a random map generator, and possibly some kind of procedural map generation and a continuous "survival" mode. The single player mode naturally comes with some simple AI enemies. Nothing special here — the game will not amount to much without these basic features. The most important goal is for the game to be fun to play, which can hopefully be achieved by enough testing and polish.

As for the optional requirements we were planning the following. Configurable keys and audio should be easy to implement - audio can be supplied through SFML (more on external libraries in the end of the documentation), and configurable keys can be implemented with simple menu selections and the like. "More intelligent AI" is somewhat vague as a requirement, but we do plan to implement more refined AI than just homing towards the player, for example. Sophisticated AI also allows us to provide AI players in multiplayer mode. Basic implementations of weapons and terrain are designed to be flexible, making it easy to add new ones. Campaign and map editors are in the planning and should not be hard to implement (not to mention that making maps by say, writing text files sounds like a terrible idea). We hope that our final map generator fulfills the criterion of being "super cool".

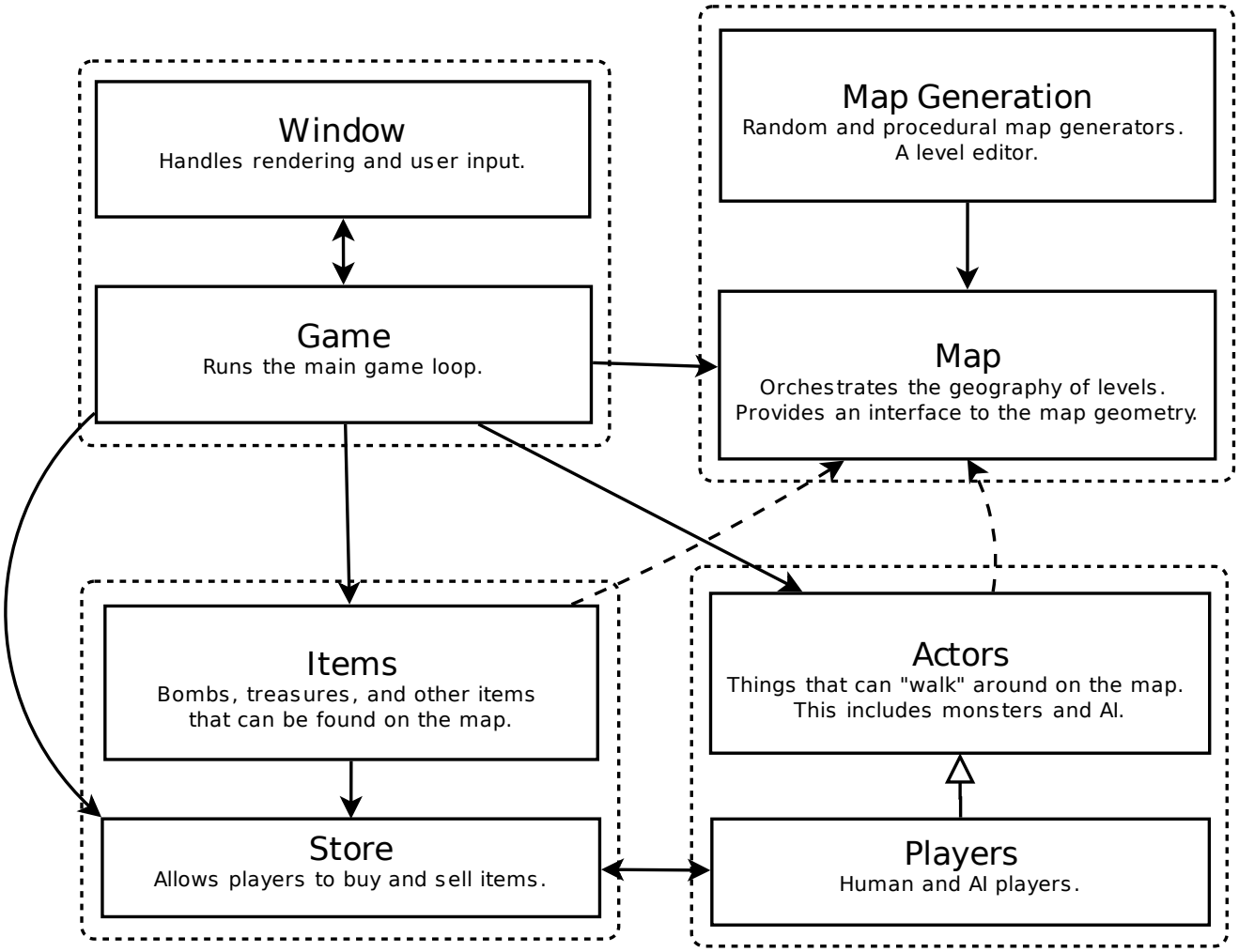
We discussed the possibility of netplay or some other type of multiplayer mode,

but decided that such features should only be considered once a solid foundation is in place. We strongly believe that a well-built set of basic features makes for good extensibility, including netplay and other large-scale improvements.

The features outlined above are the ones that have been discussed so far, and they form our starting plan. Plenty of work will be needed to refine them to excellence, though, and experience may bring to light unforeseen difficulties or new opportunities for exciting features. We recognize this and acknowledge that the specification will evolve along with the implementation.

Program architecture

The diagram below describes the general architecture of the program. The structure is pretty straightforward - the main class Game maintains the state of the game, Actors represent the moving and sentinent things in the game, Map describes the maps games are played in, Items are items to be collected and used and so forth.



The architectural design divides the the program into four quite distinct parts - the map and its generation, the actors, weapons and other items, and the overall game logic. This makes implementaion on separate parts of the program easy, and on the other hand divides logically separate things into their own sets.

In the design phase most of the problems we faced and discussed revolved around where the data is stored and how it's accessed, specifically around the way that objects in the game can request information about the map's geography — for example, when an AI player wants to know what's in the block in front of him, should that information be retrieved from the map or the block he is currently at? Should the map contain, for example, quick access to player locations or should it just find them once somebody requests them? We ended up storing most of the data for players and such in the Map object, so that map blocks don't need to contain references to other blocks or objects. Say, for instance, that an actor wants to know what the block he's facing contains. In order to get the location of that block, he would call a member function of the Map object, supplying his own location and the direction he's facing as parameters. Other possible queries include "first blocked (non-walkable) block in the given direction", "the set of blocks within distance X of the given block", and so on.

Task sharing

Areas of responsibility are divided among the developers, roughly speaking, into the ones outlined by the architectural diagram. In addition to these tasks, graphical resources need to be prepared. The division is as follows:

Miro	Maps and Generators
Henri	Items and Store
Joonas	Actors and AI
Roope	Game logic and graphics

Towards the end of the development process, some areas will probably prove more labor-intensive than others, and development tasks will overlap. The primary areas of responsibility will nevertheless continue to be the ones described above.

Efforts will be coordinated in weekly meetings and via IRC. To make things go smoothly, interfaces will be made as clear as possible. Maps provide access to different aspects of level geometry, Actors have member functions to govern their movements, Items can be picked up from the map, and the Game contains logic to make it all tick.

Testing

A test suite will be written and maintained alongside the program code, consisting of a set of unit tests for the nontrivial aspects of each component, as well as some tests that involve the central interactions between components. These tests will be run for each new iteration to ensure that new code does not break existing functionality. In addition, game mechanics will be aggressively play-tested for enjoyability — fun and games are serious business.

Schedule

Development will start with the production of a rough-but-functional prototype, which will then be iterated on to arrive at the finished game. We expect to produce this first prototype within about a week. From then on, an agile-like process will be used: weekly meetings will set targets for each week, progressing from essential components towards nice-to-have features and optional extras. The following table provides a rough roadmap for the project. The schedule is tight, but good results require hard work.

01.11 - 08.11	First working prototype (some maps, items, weapons and moving players)
09.11 - 15.11	random map generator, monsters with AI, item store, more weapon types
16.11 - 22.11	Minimum requirements fulfilled. Work started on some optional extras
23.11 - 29.11	Sweet graphics and audio
30.11 - 08.11	Bug fixes and finalization, documentation

External libraries

SFML will be used for graphics, input handling, and other tasks. At present, we foresee no need for any other external libraries.