TDDC74 - Projektspecifikation

Projektmedlemmar:
Henrik Österman henos134@student.liu.se
Erik Bäcklund Ekvall eriek286@student.liu.se

Handledare:
Johannes Schmidt johannes.schmidt@liu.se

30 maj 2015

Innehåll

1 Projektplanering

The project is supposed to result in a game-engine made specially for making 2D roleplaying games. The focus is on making tools to rapidly produce simple games.
The code will be object-oriented primarily, using multiple inheritance but functional code will also be a part of the project. As an example we intend to write our own scripting language which will be functional.

1.1 Kort projektbeskrivning

A 2D RPG game engine with tools. Examples of tools are a scripting language for making dialogue, pre-existing class structure which allows quick and easy creation of new objects etc.
We recommend looking at IceBlink Engine and Final Fantasy 4-6 for examples of the type of engine we will try to achieve.

1.2 Utvecklingsmetodik

We will work separately the majority of the time and later have follow-up meetings. In practice this will result in us working in the same room but separately so we can communicate progress as it happens. We will use git for revision control. The git repo can be found at www.github.com/zappater/riverengine/.

1.3 Grov tidplan

We will implement one version of both the graphics engine and game engine every week. By the half time meeting we will be working on version 3. Information about what the versions contain can be in the document 'Version.org'.

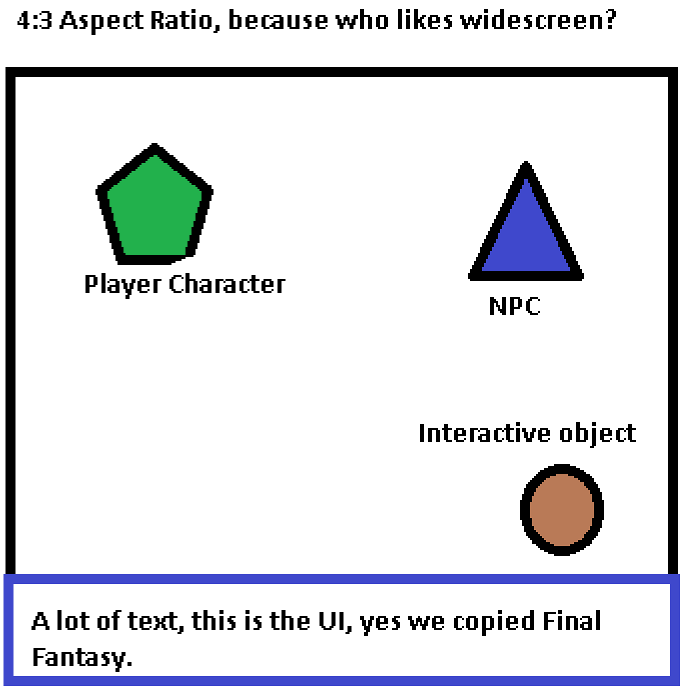
1.4 Betygsambitioner

We will only attempt to achieve the grade 3 on the project as we both already have a 5 in the course from the exams. This allows us to let other courses take more time if they need to. However we expect to achieve a 5 on the project if we aren't distracted by other courses.

2 Konceptskiss

For the developer: A lot of code, more like a library than anything else.

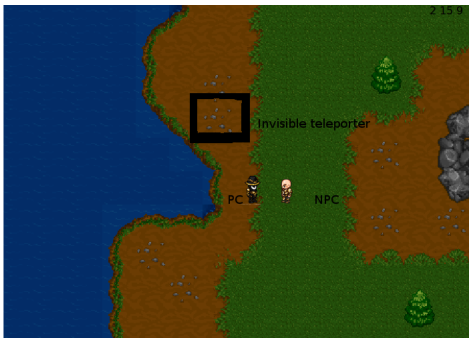
For the player: A lot like Final Fantasy, look at the picture and use your

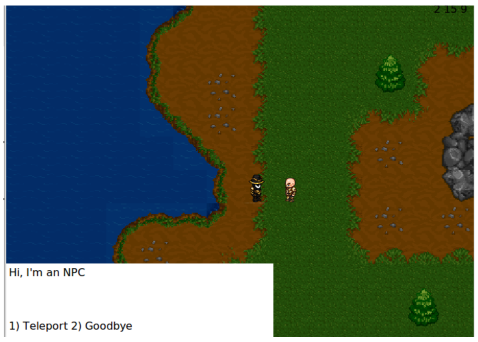

4:3 Aspect Ratio, because who likes widescreen?
Player Character
NPC
Interactive object
A lot of text, this is the UI, yes we copied Final Fantasy.

imagination.

3 Användarmanual

The final product includes two parts. A demo game, this is started by running draw.rkt and a map editor, this is started by running act-editor.rkt. In addition to this it includes a lot of small subsystems which is explained in the comments to the code.
The demo game is played by running draw.rkt. The character is then moved using the arrow keys and the world is interacted with using the spacebar. If entering a dialogue with a character the number keys 1, 2, 3 and 4 is used for choosing dialogue options. The current demo game includes a walkable world, one NPC which can be interacted with by standing next to it and looking at it then pressing the spacebar and two teleporters (without textures so they are invisible) which can also be interacted with using the spacebar.

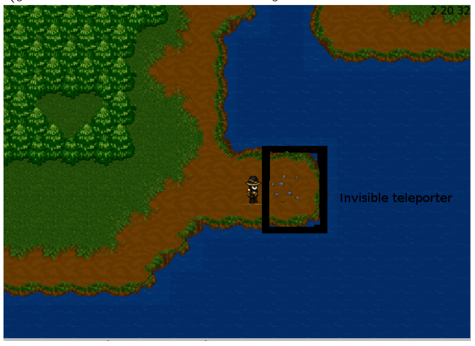




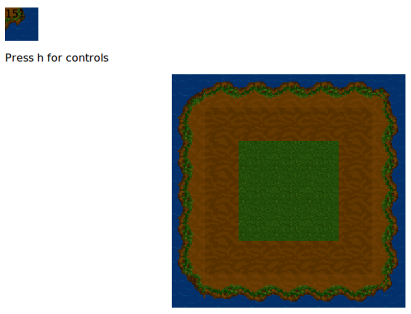
Hi, I'm an NPC

1) Teleport 2) Goodbye

The first image shows the start of the game. The second shows conversation with an NPC. And the last image shows where a second teleporter is hidden (you arrive there if you use the first teleporter).



The map editor is started by running act-editor.rkt. Press h for help and h again to return to the map editor. Simply put you place tiles using the left mouse button and remove them using the right mouse button. You scroll through textures using a and q, you change level to put the texture on with s and w. This will result in a .txt file containing the map. This is then copypasted into a correctly formated .rkt file (make sure to specify language used at the top of the document!). To get a fully running game additional code for objects, required rkt files and control scheme need to be added. This is a bit too much to explain here, this is after all a game engine not a game maker! It is made to be a framework for programming your own games within not as a tool for generating complete games without any programming. We suggest looking at Demo_game.rkt for an example.



Act-editor, a small island in the making.
the top left corner is showing the currently selected texture

3.1 Kravlista

#	Beskrivning	Prioritet
1	NPC dialogue read from file and scripted using its own language	A
2	Scripting language for making objects	A
3	Simple graphic editor	B
4	2D graphics engine	A
5	Scripting language for controlling UI	A
6	Support for animations	B

4 Implementation

4.1 Abstrakta datatyper eller klasser

Act: Contains information about what should be read in to memory. It has a type (game or UI). An act contains a n-dimensional vector (in the mathematical meaning) of levels. A game is a collection of acts.
Level: A level is a collection of pointers to objects which are in the same interactionlevel and will be drawn in the same layer by the graphics engine. The difference between interactionlevel and layer is that a interactionlevel determines what the player can interact with while a layer determines when the graphics engine draws the object.
Classes: Items, NPCs, Player Character, World Objects, Sprites, Background. All of these with their own subclasses.
Dialogue will be its own type of ADT. In essence it will be a graph implemented using linked-lists and contained in one or several textfiles.

4.2 Testning

Unit tests will be created as the engine is developed, we are not yet able to specify unit tests.
We will test the system by making a simple techdemo in it. It will contain 1 PC, 2 NPCs with dialogue, 1 interactive world object, 1 item. It will be made in 2 acts.

4.3 Beskrivning av implementationen

For a lot more expansive explanations see the code. While it is a 2D game engine, the world is built in three dimensions. A character interacts on a 2D surface called a level, it can only interact with things on its own level. A level is a mutable datastructure built up as a matrix with each cell containing one object and the coordinates of the cell, if the cell is occupied. An empty cell contains an empty list '() and the coordinates of the cell.
These levels are then layered on top of each other into a three dimensional datastructure. This is done by placing them into a list (immutable). It is immutable as once a complete world has been built the number of levels should never change while the content in each level may change. This layering of levels is done in a datastructure called an act, it is essentially a map of the game but as map has other uses in programming I will from now on only use the term act. A map is implemented as an object and functions essentially as the game server. It controls where everything is, it decides where everything is and adds, moves and removes everything from the game world.
A cell can only be occupied by one object at a time. As such this has the interesting implicating that collision detection is automatically implemented. If you want to prevent a character from being able to walk somewhere just place an object in that cell. This is also why the world is constructed as a three dimensional world. If you want to have grass that the character can walk on you need to put the grass on a level lower than that which the character is on. This also allows bridges to be built by for example making a world with four levels. On level one you put a bit of ground, on level two nothing and on level three you put the bridge and on level four nothing. Now the character will be able to walk under the bridge when on level two and on top of the bridge when on level four. Moving the character between levels is easiest done with the already implemented teleport object, which when used tells the act to move the character to a prespecified place. I hope this illustrates why the world is constructed as it is.

5 Utvärderingar och erfarenheter

Planning the project and datastructure went very well. It works almost exactly as we had imagined with a lot of other benefits which we didn't consider (such as automatic collison detection). What went bad is that we focused on a lot of other courses and did not have the time to implement everything we wanted to. However we still consider the product to be good enough and easy to extend in order to fullfill our original vision.
We probably spent to little time on this project, but at the same time we had a lot of other important things to do. It is hard to say. The work load was even until the final weeks when implementation of the graphics required a lot more than expected, a part which was primarily on Eriks shoulders. It is hard to say what should have been done in order to improve the workload balance as the graphics couldn't really be started until the basic datastructure existed and by then a lot of time had already passed by. With a few more weeks of project time the workload would probably have ended up being balanced once again.
The specification has not really been used. What was useful was all the preparation we did before it but we had to do it anyway. What has been useful is a lot of documentation around the specification such as Versions.org, a document where we in a more flexible way wrote down what, when and how things should be implemented. We worked as we imagined in the beginning then things broke down towards the end because other courses took up time. When it worked it worked well, when it didn't work things didn't work well. The hardest problem has been to find time when we are both available and not focused on other courses. One of the biggest lessons has been to create proper documentation of all code as it is created and take the time to visualize datastructures. It is much easier to comment and document code while it is being written and if it is already commented it is much easier to change the code. Ending the project with commenting code for 15h isn't the best thing to do.
Figuring out how to do things when the datastructure is something very abstract can be very hard to do and in the beginning we had to draw our datastructure on a whiteboard several times to understand where we were in it. Create a lot of help functions if only to make the code easier to understand even if it is something very simple.

6 Tidrapportering

Can be found in separate spreadsheet.