Samuel Williams
Assignment 1
COSC416 - Mobile Graphics
16 August 2010

SNAILS

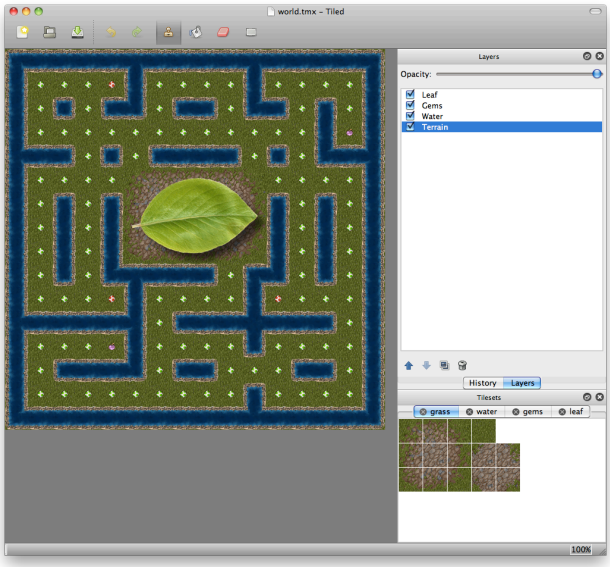
Snails is a simple 2D game similar to PacMan. The player controls an animated character to collect eggs around a 2D map. There are several snails randomly traveling around the map and these will cause the player to reset to the starting position.

Example of the game running on the JavaME simulator.

Control the animated character using the directional control pad. Collect an egg by walking over it. Avoid contact with the snails. Each egg contributes to the score which is displayed onscreen.

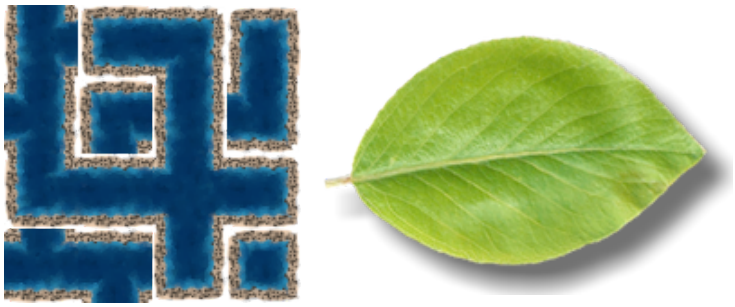
The game makes extensive use of the JavaME LCDUI classes including the LayerManager, TiledLayer, GameCanvas and Sprite classes. The GameWorld class uses several tiled layers for representing the world. The main GameEntity class provides behavior for both the human player and computer snails (via a sub-class GameMonster). Thus, it is easy to add multiple monsters to the game world, and by default there are 6 snails.

The map was put together using Tiled Map Editor, a fairly powerful map editing tool. The map data was hard coded into the application to reduce the amount of development work required.



The Tiled Map Editor displaying the game world.

Many different sprites were used. Most of them were copied from online sprite websites, however the leaf and water tiles were created by myself.



The Tiled Map Editor displaying the game world.

The collision detection for the player character is done by checking against the water layer. This basic collision detection also applies to snails, but they have a more advanced automatic movement to ensure that they vary their path when they have a choice.

Resources Used

I. Snail Pixel Art (http://wiki.themanaworld.org/index.php/User:Fother/Pixel_Art)

II. Yoshi Pixel Art (http://darkut.free.fr/Resources/spritesheets.htm)

III. Grass and Dirt Pixel Art (http://www.lostgarden.com/2006/07/more-free-game-graphics.html)

ROBOTS

Robots is a simple 3D simulation where the user can control a robot walking in a circle. The player can view from and navigate with a global camera or switch to a robot tracking camera.

Example of the game running on the JavaME simulator.

Control the global camera by using the directional control pad. Animate the robot by pressing button A. Select the global camera by pressing button D or select the robot tracking camera by pressing button B.

The game makes extensive use of the JavaME M3G classes. The GameWorld class uses a scene graph to manage all game objects. This includes many box meshes and tree sprites randomly distributed around the world, along with a subdivided floor mesh and distant sky sprites.

There are two lights, a main light in the center of the floor, and a spotlight centered above the robot. The spotlight moves with the robot.

The main camera can be moved in any direction using the directional control pad, and the tracking camera views the robot from behind. The tracking camera follows the robot as it moves.

Textures

Several textures are used to increase the realism of the scene. Some textures use alpha blending for accurate compositing.

Several of the textures used in the game world.

Robot Composition

The robot is build from a series of textured boxes. These boxes are transformed and then added into appropriate scene nodes in order to produce a simple robot model.

Robot Animation

Animation of the robot is done using quaternions. The values for the rotations were generated offline by another program and hard coded.

Rotation	**Quaternion**
45°	{0.382683f, 0, 0, 0.92388f}
-45°	{-0.382683f, 0, 0, 0.92388f}
0	{0, 0, 0, 1}

These rotations were used to generate keyframe animation for left and right side movement. The keyframe sequence uses spherical linear interpolation. The legs and arms are assigned alternative sequences so that they move in opposite directions. This produces a marching style animation.

Left Side	**Right Side**
0°	0°
-45°	45°
0°	0°
45°	-45°

The robot walks in a circle and this is also controlled using a keyframe sequence of rotations, using a set of quaternions representing an entire 360° rotation around the origin.

Floor Generation

In order to increase the accuracy of the lighting, the floor is generated using triangle strips. The size of the subdivision can be changed and the triangle strips will be generated appropriately. This allows the accuracy of the lighting to be adjusted depending on performance requirements.

Moon and Stars

The background moon and stars are generated and placed randomly, with the moon always within the initial global camera view. They are added to the scene graph using Sprite3D instances at a great distance from the origin to simulate parallax.

Bugs

Generally, the program works well. However, I have noticed some issues with the main light and the floor. Sometimes there are visible artifacts. I have checked and double checked the floor generation code and can find no fault, thus I have to assume there might be a fault with M3G.

Resources Used

I. Crate Texture (http://stormvisions.com/pages/insert/burt-art.php)

II. Pine Tree (http://digitalholeinone.com/trees-textures-materials/3ds-materials-textures.html)

III. Grass Texture (http://en.cze.cz/Textures)

IV. Moon Texture (http://people.dbq.edu/students/jgarien/Webquest.htm)