Justin Strandburg
W01099158
CS321
WWU Spring 2014
Pygame Project: Bob the Blob
Game Manual

Table of Contents:

User Guide:

Summary:

Bob the Blob is a hybrid physics/platformer game. The goal of the game is to move Bob the Blob from his starting location to the goal location by interacting with various physics objects. The game is written in python, and uses the pygame and pymunk libraries to do the graphics/sound/input and physics simulation respectively.

Game Objects:

- The player: Bob the blob
- Enemies: Killer cyborg sheep bent on the extermination of the entire Blob race
- Platforms: Static pieces of land that the player can land on. They can be made of several different materials:
    - Wood: high friction and low bounce
    - Slime: medium friction and high bounce
    - Ice: low friction and low bounce
    - Lava: If you touch this you die
- Spinners: Dynamic platforms that rotate about their center. Spinners can be made of the same material as platforms (except laval). These operate in three different modes:
    - draggable: these can be repositioned by the player if they are clicked on
    - free: these will respond to physical interactions with the player and enemies
    - fixed: these rotate at a fixed angular velocity
- Gravity wells: Circular zones that either attract or repel the player. While the player is in a gravity well they are not subject to normal gravity, so gravity wells can be used to perform sweet orbital maneuvers. If you get stuck in the center of a gravity well it will temporarily “turn off” giving the player a chance to escape.
- Force fields: Rectangular areas that push the player in a specified direction when the player is within their bounds
- Magnets: Circular objects that the player can stick to.
- Sparkly things: Sparkly things are your goals. If you touch a sparkly thing you win the level.

Controls:

You control Bob by launching him through the air. You launch him by holding down the right mouse button to charge up, then when you release the button Bob will launch in the direction of the mouse.

Besides launching Bob, the only other thing in the environment you can directly control is draggable spinners. By left clicking on them and dragging the mouse you can rotate draggable spinners. If you rotate a spinner while Bob is sitting on it you can launch Bob through the air at much higher speeds than he can launch himself.

In addition there are the following keyboard controls:
Q - Restart the current level
F2 - Switch between standard graphics and primitive graphics
Z - Teleport Bob to the mouse location like a dirty cheater
ESC - Pause the game
F4 - Take a screen shot (will be saved to screen.png)

Running The Game:

Most of the source files are located in the src/ folder. There is a single python file blob.py in the root directory. Executing blob.py will run the game.

In-Game Tutorial:

There is a short illustrated tutorial available from the main menu.

Code Documentation

animation.py:

Helper classes to load animations and play animations from a custom animation file format I made up.

Classes:

AnimatedImage - Class that loads and manages an animated image from a simple file format that I made up. Provides a method that creates an AnimationHandle for the given animation.
AnimationHandle - Class that provides a cursor into an AnimatedImage without duplicating the frames or timing information so the same animation can be used in multiple locations asynchronously
AnimatedSprite - An extensions of the pygame.Sprite class that provides an interface for loading and cycling multiple animations.

framework.py:

Generic framework that provides a foundation that could be used to write any game. There is no game specific logic in this module.
classes:
SettingsManager - A class that parses a configuration file and provides an interface to request variables from the file with default values if the variable is not found.
ResourceManager - A class that loads a list of resources with tags from a file, and provides an interface to request resources by tag. This class implements a lazy loading scheme where none of the resources are actually loaded until the first time they are requested.
Activity - A base class for the various different screens/activities that are implemented in the game. Provides an interface for activity creation/deletion, event handling, and variable timestep updates. This class was designed to be similar to the Activity model used in the Android framework.
GameController - A class that does event handling and activity management.

gameplay.py:

Game specific code is all found in here

Classes:

SegmentRenderer - Helper class that handles platform rendering, which is a little complicated.
SpinnerRenderer - Help class that handles spinner rendering. Since this requires lots of image rotations, the results are cached to improve performance.
GameplayActivity - Activity that runs the core gameplay

Functions:

get_attrib - Helper function to help extract attributes from and xml file, used in loading levels

menu.py:

Base classes for menu implementation. No specific menus are in here

Classes:

MenuWidget - A base calss for all menu items
TextWidget - A simple text menu item
TextButtonWidget - A selectable/clickable menu item
MenuActivity - A base class for all menu activities. Handles the management of widgets and responding to input events.

rungame.py:

The main control loop and the specific menu implementations are all in here

Classes:

MainMenuActivity - The Activity for the main menu
TutorialActivity - The Activity for the illustrated tutorial
LevelSelectMenu - The Activity for the level select menu

Functions:

main - Runs the main control loop.

Acknowledgements

Patform Textures:
wood: http://www.texturex.com/Wood-Textures/red+wood+texture+grain+natural+wooden+paneling+surface+photo+wallpaper.jpg.php
slime: http://1.bp.blogspot.com/-jsbISkx4slM/UGLA_lMoF_I/AAAAAAAABX4/cRb6sUQEBlU/s1600/green-slime(1).jpg
ice: http://honey-stock.deviantart.com/art/ice-texture-for-oss-funfair-167357464
lava: http://farm4.static.flickr.com/3371/3600528652_b49b9e9468_o.png

Spinner/Enemy Textures + Level background:
SMBX Graphics Pack: http://rpgmaker.net/engines/smbx/utilities/20/

All sound effects:
freesound.org

All other art/sprites were hand drawn by me using GIMP

About the Author

What first got me into programming was the idea of working on games. As such I spent quite a great deal of time working on games in high school. Unfortunately since my programming knowledge was entirely self taught and I was working in C++ I was never able to actually finish a game before the burden of bug hunting and maintaining a largish code base in C++ sapped my enthusiasm for the project. This is the first project I have ever done in python, the first game I've ever used a physics engine for, and the first game I've actually completed. I'm pretty proud of it.