Birdman Report

Alex Zhang Jing Quan 100101666

*External Libraries*

* freeglut-2.8.1
* glew-1.12.0
* ~~FTGL (laggy trash)~~
* freetype-2.5.5
* SOIL (July 7, 2008)
* glm-0.9.6.1

*Art Credits*

License: CC 3.0

* Zeyu Ren <http://opengameart.org/content/backgrounds-for-2d-platformers>
* Zack Alvarado <http://opengameart.org/content/platformer-tilesets>

License: Public Domain CC 0.0

* Jason-Em <http://opengameart.org/content/classic-hero-and-baddies-pack>

License: Unspecified

* Coin <https://projectwerk.webontwerp.khleuven.be/projects/melonjshtml5spel/wiki>

License: Freeware

* TTFont from <http://dsg4.com/04/extra/bitmap/>

*Code Credits*

OpenGL Lesson 43: TrueType Fonts With FreeType

<http://nehe.gamedev.net/tutorial/freetype_fonts_in_opengl/24001/>

Authors Name: Sven Olsen

Birdman is a basic 2D game engine which features animated sprites, tilemapping, chase camera, time tracking and simple collision detection and response. It is also a platformer game which utilizes the capabilities of the engine to create a vibrant and interactive world.

Core classes

*main.cpp*  
Initializes engine components and glut.

idle() – The main loop of our game, game logic, physics and calculations goes here.  
display() – Opengl calls goes here.

dt (second taken to render a frame) is tracked so game is not framerate dependent.

*camera.cpp*  
Builds the projection and view matrices, and handles smooth panning in the game world. Used glm to build the projection and view matrices because, well it is the modern way, also in case I need them for shaders.

moveTo(vec3 position) – Move camera to specified position smoothly.  
Simply take difference of destPos and camPos and add resulting vector to camera’s velocity until destination is reached.

Camera can be panned smoothly but unused currently.

*cursor.cpp* (extends sprite)Draw a cursor sprite, and store its position in local (window) and world coordinates.

*timer.cpp*Make a timer object that ticks every specified seconds, for specified duration, to be used in the update loop. All timers are stored in a static list so they can be updated all at once, and set for removal.

*Sprite.cpp*Any object that needs to be displayed in world is a sprite. Each sprite is basically an alpha blended textured quad with position and size, and texture loading is done by SOIL. You can change basic properties of the graphics including zOrder, color, alpha, visibility and horizontal flip.

setupAnimation(…) – Setup spritesheet animation, with variables including size of each frame, intervals between frames and the start and end frames to loop. Map the quad’s texture coordinates to each frame accordingly by first calculating how many frames are in the sprite sheet, then map to 0.0 - 1.0 range.

setupCollision(…) – Setup AABB collision detection. Variables needed are the bounding rectangle size, and an optional position offset. Next a boundingRect (vector4) coordinate is build local to the sprite’s position.

isCollidingWith(Sprite\* sprite) – Check if sprite’s bounding box is colliding with another sprite’s if their distance is smaller than their half bounding box’s size, in both axis.

By default a quad is made and added to a list of vertices and texture coordinates, and drawn using glDrawArrays. More quads can be easily drawn by populating those lists.

*tilemap.cpp* (extends sprite)  
Each tile in the game world is build accordingly to the csv data sheet. e.g.   
 c, c, c, ,ib,  
 c, c, c, ,19,  
 c, c, c, ,35,  
ib,ib,ib, ,35,  
20,21,21,22,51,  
Position of each value, which are separated by commas, is relative to the tile’s position in the game world, multiplied by the tilesize. Positive values are the frame index of the tilemap’s spritesheet which determines the collidable terrain of the game world and if it is negative, no collision is checked in the later phase, but the same positive frame index is used to display the tile. Vertices and texcoords are build for that tile and stored in two lists respectively. If the value is a string, it is considered an intractable game item and an Item struct with position and string value is stored in a list of Item.

getCoordAtPos(vec3 position) – Finds the tilemap 2d index at the position in the game world. Used in conjunction with

getValueAtCoord(int x, int y) – Returns the value of the tilemap at specified index. Used for collision detection of player and Npc sprites.

*character.cpp* (extends sprite)  
Any character or object that is affected by tilemap collision and gravity uses this class. Implements basic character state such as moving left or right, jumping, on platform or is dead. During each iteration in the update loop, collision with the tilemap is checked in all 4 direction, based on the character’s bounding box and its next possible position. If it hits a wall in the x axis, set the position so that the bounding box just grazes the wall. For the y axis, if character is colliding with platform by standing on it, gravity is disabled and its y position set to just graze the platform. Collision is done this way so there is no phasing through walls or jerky movements unless player is moving at very high speed. moveSpeed and jumpSpeed can be specified for the character.

*guy.cpp* (extends character)  
The player’s character, which only handles state changes depending on input and updates the animation frames accordingly. If damage is taken, flash and knockback. Bounce() is used when player stomped an npc. Punching spawns a projectile and kicking increases movement speed. Score such as coins collected and npcs killed is tracked. It only includes a function to reset the states when the game restarts.

*npc.cpp* (extends character)  
Setup animation frame and movement speed and basic movement logic for monster red and blue. Same concept as guy class.

scene.cpp  
Manages what goes into this current scene, and what is destroyed when scene ends. If further levels are added this will come into use. During load(), all the objects, camera and fonts used in the game are initialized, and the scene is reset, which clears all items in the map (coins, npcs) and create new ones at defined locations in the tilemap. Camera and player is reset and positions and the game can begin. In the update loop, player and npc collision logic is handled, including collecting coins, tracking player damage and stomping npcs.