Upstream
REMAKE OF WINTERBELLS

QUY NGUYEN, JONATHAN DINH, JOHN HARGREAVES, KEVIN JENKIN
CMPS 335 | APRIL 15, 2016
## TABLE OF CONTENTS

GAME OVERVIEW .................................................................................. 2

1 REQUIREMENTS ................................................................................. 2
   1.1 SOURCE CONTROL ........................................................................ 2
   1.2 FRAMEWORKS ............................................................................ 2
   1.3 GOALS ....................................................................................... 2

2 DESIGN ............................................................................................. 3
   2.1 GAME INTRODUCTION ................................................................. 3
   2.2 GAME MENUS & PROMPTS ....................................................... 3
   2.3 INPUT & GAMEPLAY ................................................................. 4
   2.4 ANIMATION .............................................................................. 4
   2.5 PHYSICS ................................................................................ 6
   2.6 HIGHSCORES .......................................................................... 8
   2.7 SOUND .................................................................................. 8
   2.8 FLOWCHART ......................................................................... 9

3 IMPLEMENTATION ........................................................................... 9
   3.1 MODEL .................................................................................. 9
   3.2 CODING STANDARDS .............................................................. 9
   3.3 DOCUMENTATION .................................................................... 9

4 TESTING .......................................................................................... 10
   4.1 UNIT TEST ........................................................................... 10
   4.2 STRESS TEST ....................................................................... 10
Game Overview:

Upstream is a remake version of the famous and fun game created by Ferry Halim called Winterbells. The original game can be play for free at this website here:

http://www.ferryhalim.com/orisinal/g3/bells.htm

Winterbells is a single player game where the player controls a rabbit movement with a mouse and tries to jump on bells falling from the sky without falling to the ground. The longer the player successfully jumps on each bell, the higher the score will be. The bell will disappear as the rabbit jumps on it. The bell will also reduce in size the longer the player survives without falling, which makes the game more challenging and fun. The goal of the game is to survive the longest and gets the highest score.

We will attempt to make a game similar to Winterbells with added features called Upstream. The basic game concept is the same but our game is about a frog jumping upstream against a flowing river. The river will have Lily pads and logs flowing along it. The objective of the frog is to jump upstream opposite to the current of the river with the help of the lily pads and logs without falling into the river.

1. Game Requirements:

1.1. Source Control

The source control we used for storing our source code was GitHub. We created an organization repository, where members can be added and given permission to alter the source code of the project. Each member is able to clone the GitHub repository, work on the code, and push it to the master branch. Our instructor can access the project source code by cloning the repository URL provided at:

https://github.com/upstream335/upstream

Each student will have a separate .cpp file named after the student’s first name and last name initial. The .cpp will contain the source code written by each student throughout the software development phase.

1.2. Frameworks

The frameworks we used for our project includes the rainforest and battleship provided by our instructor. The rainforest’s source code was stripped down to only contain the essential functions needed for the project. The button features in the battleship source code were integrated into the project.

1.3. Goals

The project initial goals were:

- Game menu
- Animated background implementation
- Animated sprite implementation
- Obstacle objects implementation
- Physics implementation
  - Gravity
  - Collision detection
  - Random factors
  - Motion
- Sound implementation
- Attempt to:
  - Multiplayer
  - Web page for updating high scores/leaderboard

Goals met & added features:

- Everything listed in the initial goals except for multiplayer.
2. Game Design:

2.1 Game introduction

When the game executes, the game will first enter the introduction screen. The introduction screen will render the main menu displaying the name of the game, background theme, sound effects, selection buttons, and an animated frog object as a preview of the game. The main menu is launched at the beginning of the game and can also be accessed in-game. Here is a snapshot:

![Game Introduction Screen](image)

2.2 Game menus & prompts

The game menus and prompts are implemented as buttons with texture mapping using openGL. There are a total of fifteen buttons implemented in-game. Five buttons in the main menu, five buttons in the sub menu, three buttons in the game-over menu, and two buttons in-game.

- **Main menu buttons**: play, difficulty, score, credits, exit
- **Sub menu buttons**: resume, sound/mute, restart, main menu, exit
- **Game-over buttons**: play, main menu, exit
- **In-game buttons**: sound, help

Each button has its own functionality in the game. The buttons are activated when mouse is hovered over the button and left mouse clicked.
### 2.3 Inputs & gameplay

The main inputs for gameplay is the mouse and ESC key but other keypresses are embedded for demonstration purposes. Left clicked for making the frog jump and once the jump is in the air, movement of the mouse cursor controls the frogs’ movements. Right clicked for rocket mode, an additional feature to the game. Other functional keypresses include:

- **t** – demo mode
- **i** – reset game
- **o** – initiate game-over menu
- **p** or **ESC** – initiate pause (sub-menu)
- **h** – help menu page
- **j** – lily pads oscillates
- **k** – stress test (generate more lily pads)
- **s** – swarms of flies
- **b** – spawn boss that shoots out a single bullet

### 2.4 Animation

For the game Upstream we rely on openGL, and sprite based animation to create our game world. OpenGL (Open Graphics Library) is a cross-platform, hardware-accelerated, language-independent, industrial standard API for producing 3D (including 2D) graphics. Modern computers have dedicated GPU (Graphics Processing Unit) with its own memory to speed up graphics rendering. OpenGL is the software interface to graphics hardware. In other words, OpenGL graphic rendering commands issued by your applications could be directed to the graphic hardware and accelerated.

We use 2 sets of libraries in our OpenGL programs:

1. **Core OpenGL (GL):** consists of hundreds of commands, which begin with a prefix "gl" (e.g., `glColor`, `glVertex`, `glTranslate`, `glRotate`). The Core OpenGL models an object via a set of geometric primitives such as point, line and polygon.
2. **OpenGL Utility Library (GLU):** built on-top of the core OpenGL to provide important utilities (such as setting camera view and projection) and more building models (such as quadric surfaces and polygon tessellation). GLU commands start with a prefix "glu" (e.g., `gluLookAt`, `gluPerspective`).

In computer graphics, a sprite is a two-dimensional bitmap that is integrated into a larger scene. Basically it is a 2D image used together with some data to position it in a larger world using a position, a rotation angle and a two dimensional size. So, sprites are the render-able image/texture objects we use in a 2D game. We use the files ppm.h and ppm.cpp to load image files for use as textures that will be displayed in our sprites. First though since our images are saved as png image types to allow compression and save space we have to convert the images to ppm which is done in the function `Ppmimage* get_image ( std::string filename )`
In order to give our characters and the environment a more real appearance, each sprite can actually be represented by several different images stored as textures. Rotating through these different images gives static objects the appearance of motion, a common animation technique. These separate images are referred to as frames. For example, here are 10 images that represent the frog moving from left to right:

And here’s the frog jumping frames

These images were created in Photoshop and saved as individual PNG images. Since the original framework (rainforest) we started with, used black as the alpha color the background was set to black before saving and then in the game, black was rendered transparently allowing previously rendered layers to be visible ‘behind’ a new object. Other techniques used in the game involved drawing lines using openGL applying colors to layers and rendering text using libggfonts.a. We even created our own bitmapped frog font using techniques observed in the example fonts.h and fonts.cpp provided.

Website: http://www.cs.csuh.edu/~jhargreaves/upstream/scores.php

The website uses a simple CSS to create a uniform ‘look and feel’ between its pages. The primary functionality is in the scores.php page where php code is used to do the following:

- Listen for a specific tagged string to be received via form indicating a new score
- Read in top scores from highscores.xml
- Sort scores based on values (Hard and medium game modes are weighted higher than easy)
- If the list is longer than 20 players drop the lowest scores
- Write out the lowest score in each mode to lowscores.txt (these are minimum scores to beat in game to get on scoreboard)
- Write the new list of players back out to highscores.xml
- Display the list in a nicely formatted table designated by the CSS
  The website also makes use of a lot of JavaScript combined with the images of the frog created for the game. These images are used to create animations of the frog jumping on each page of the website and to create a simple game ‘Frog Click’ where the player gets points for clicking on as many frogs as they can.
In the game code, C++ is used to read in the scores to beat from lowscores.txt before saving a player's name, score and name and sending to the website. An additional library tinyXML2 is used to parse highscores.xml and these scores are saved as strings to be displayed on the high scores screen.

2.5 Physics

Frogs:
- Frog object is controlled by the player. Frog interacts with all the other objects in game.
  - If the frog is within the area of the alligator mouth, then the frog will be eaten.
  - If the frog lands on top of the alligator's back, then the frog's velocity is up.
  - If the frog lands on the log, then the frog's velocity is up.
  - If the frog is near the radius of a fly or swarm of flies, then the frog eats the flies and gaining bonus points.
  - If the frog is within the radius of the golden turtle, then a huge bonus points will be added to the high score.
  - If the frog reaches 2,000 points, then the boss will spawn.
  - If the frog hits the bullets, the frog will die.
  - If the frog lands on or pass through a lilt pads, the frog's velocity is up and gain points.

Alligator:
- Alligator moves from right to left and appearing randomly throughout the game.
  - If the alligator is landed on by the frog, then the alligator will submerge and resurface the water.
  - If the alligator hits the log, then the alligator will bounce back avoiding the collision.
Logs:
The logs will appear randomly throughout the game and moves from top to bottom. The log will interact with the frog and the alligator but will not change or be affected by other objects.

Flies:
A single large fly will randomly spawn but a swarm of flies will randomly spawn but follows the frog position.
❖ If flies are near the frog's mouth, then the flies will be eaten and disappear.

Turtle:
Once in a while, a golden shell turtle will randomly appear. The turtle swims across the river from left to right.
❖ If golden shell turtle is near the frog, then the golden shell will disappear.

Boss:
The boss spawn near the top and moves back and forth horizontally. The boss only appears when the frog gains 2,000 points. The boss cannot be killed but will disappear if the game reset.

Lily pads:
The lily pads are placed in a linked list structure, similarly to that of the raindrops in the rainforest program. The spawning point for the lily pads is set 30 pixels above the max height of the game window. Three different textures are mapped in a sorted order to animate the movement of the lily pads. The lily pads will start spawning as soon as the game begins, with each new lily pad created after 35 frames has elapsed.
As the frog object moves upstream, the lily pads positions are moved downward to create a sense of the frog moving upward. When this happens, we have to increase the spawn rate of the lily pads to match to original spawn rate, otherwise the distance between each individual lily pads will be too large to perform any meaningful jump.
The lily pads will resize base on the score. This means as the player accumulate more and more points, the game will try to increase its difficulty by reducing the size of the lily pads. The hitbox of each lily pad is also being adjust to properly mirror its size.

2.6 High-scores
Having your name displayed when a high score is achieved is the sole reason anyone wants to play any game. With that in mind, our game implements a system to keep track of the high scores on the server, and allows players to input their name if their score reaches the top 20 score. When the game over state is reached, the game will check the player's score and compare it to the lowest high score on the website. This is done by using the httpget functions, and outputting the list of scores into a text file for checking. If the score is eligible for being in the top 10, the game will then display a high score box for the user to input his or her name.

Since we were not permit to use a more advanced library, getting user input for player's name was done by recording keystroke events and concatenate the appropriate characters to the string. We started out with an empty character string and append the '|' character to the end of the string to simulate the text cursor. A function was set up to record the keystroke from a-z, each time a keystroke is hit, we remove the '|' character, concatenate the appropriate keystroke character to the string, and then place the '|' character back at the end. If the user does not enter anything, a default name is
placed and the score is sent to the server. Currently, due to the limited knowledge of the X11’s key
codes, the function only checks for lower case characters and number.

2.7 Sound

For the sound in our game we used OpenAL, which stands for Open Audio Library. OpenAL is designed
similar to OpenGL in its coding styles and syntax. However, it differs because it is not operating system
specific, it is a cross-platform sound API. OpenAL allows the programmer to specify objects and
operations in producing audio output.

For the sound to work in a program, you must initialize OpenAL and initialize each sound
effect to a source and buffer. In order to use each sound effect throughout the duration of the program
an array of buffers and sources must be stored at the beginning. The buffer holds the sound
information and the source refers to the sound. The source is stored in the buffer. Once it's stored you
are able to edit how the sound is outputted. For instance, the background sound file was really loud
and we didn't want it to overpower the other sounds so it was played with a 0.1f gain value. You are
also able to change the pitch of a sound and set it to looping if wanted.

On terminating the program there should be a cleanup function for OpenAL. Its job is to
delete the sources, delete the buffers, get the active context, get the device for the context, destroy
both the context and device, and finally exit OpenAL. If this is not done, the operating system cleans
it up for you and outputs a warning telling you it did so.

CONNECTING TO THE WEBSITE

We used a modified version of lab3, httpget, to connect to the web-site’s php file. We passed
3 parameters into the modified function to send to the website; player name, score, and difficulty.
Once the input is passed the function connects to the web server. The web server then connects to
Sleipnir, which then passes the input to the php file.
2.8 Flowcharts

![Flowchart Image]

3. Implementation

3.1. Model
The model we try to follow is the waterfall model.

3.2. Coding standards
The coding standards every member set was the K & R coding style. Comments where is needed and entails the overall description of the functionality rather than the specifics. No text exceeding the 80 column mark to prevent text wrapping.

3.3. Documentation
Progressed are tracked and documented every week. Here are some of the snapshots of the progress:
4. Testing

4.1 Unit Test
A form of testing that emerged during our coding was the ‘demo mode’. Originally written just to move the frog character around randomly while a player wasn’t actively playing the game, it also proved a valuable tool to reveal flaws in collision, sound, and other aspects of the game as it could be left running for extended periods to ensure actions we didn’t anticipate, wouldn’t cause unexpected results.

Other testing that was performed was play-testing by team members and their friends and family members to ensure that not only did the code perform properly but that the game had ‘fun appeal’ and elements that would make people want to play it. We also found it beneficial to test on a wide variety of computers as there were issues such as the top of the screen not displaying when on a laptop with smaller default resolution and sound or fonts not functioning on machines without necessary libraries installed. Finally testing was performed in ‘offline mode’ to ensure errors occurring when the game could not communicate with the website would not prevent players from enjoying the game.

4.2 Stress Test
One early feature implemented into the program was stress testing to see how many lily pads the game can spawn before any noticeable performance issue were exhibited. We set to have 10 lily pads spawn per frame, as well as reducing the spawn rate to the lowest value possible. While performing the stress test for the first time, we were able to quickly find a massive memory leak in our game. At one point, the program was consuming roughly 100 megabytes of memory per minute of playing time. The reason was due to the fact that we were creating a new texture map for each individual lily pad, and did not properly delete the texture after deallocating the lily pad. We were able to quickly pinpoint down the cause, and fixed the issue. By creating a single structure to hold the textures, we can now reuse the same textures for all the lily pads instead of creating new textures for each lily pad individually.

While the stress test is enabled, we display the FPS counter and the number of lily pads on screen, and found out that when there are around roughly 3000 lily pads, the fps started dropping from 60 frames to 30 frames per second. We thought that this was a good threshold and that our game is somewhat decently optimized.