CS 214 Homework 5

#1 Victory Royale

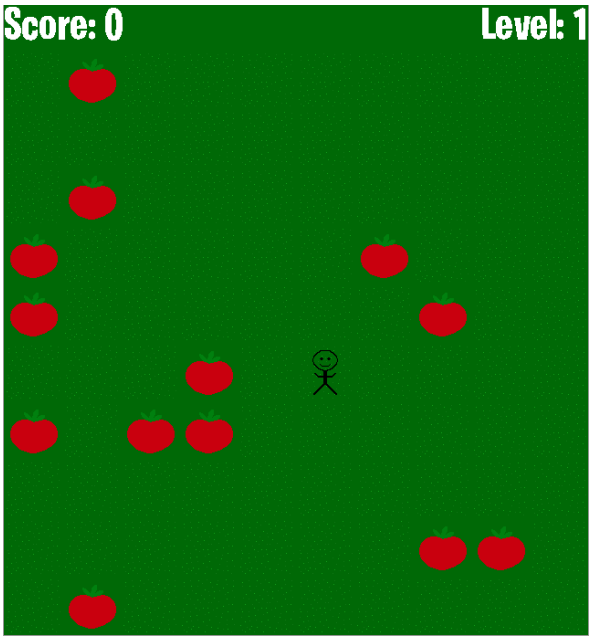
Fall 2021

Introduction

Rutgers esports needs new pro Fortnite gamers! To be a pro, first you must learn how to wipe out Tomato Town. We've provided a simulated Tomato Town you can practice in.

However, teamwork is crucial in Fortnite, so it'd be nice if our simulator supported multiple players. Your task in this project is to extend this single-player game to a multiplayer version.

A single-player version of the client is given to you. Upon starting it, you should see a field with some (Rutgers) tomatoes:



You can use the arrow or WASD keys to move your character and collect the tomatoes! Once all are collected you move to the next level. Hit ESC or Q to exit the game.

Single-player client

The client is implemented using the Simple DirectMedia Layer (SDL) library and addons for handling images and font files. For documentation, see:

- SDL wiki: https://wiki.libsdl.org/
- Images library: https://www.libsdl.org/projects/SDL_image/docs/index.html
- TTF library: https://www.libsdl.org/projects/SDL_ttf/docs/index.html

Each level consists of a 10×10 grid of tiles, which are randomly assigned to be either grass or a tomato. Once the player has collected all tomatoes, they move to the next level, and the grid is randomly repopulated with more tomatoes. (Currently there is no real difference (in difficulty, etc.) between levels.)

The `client.c` file contains the source for the client. It also uses images and fonts in the `resources` directory. You're free to modify the client as you wish.

Tiles (for grass or tomatoes) are 64×64 PNG images. The window size is hard-coded to a non-resizable 640×690 (640 plus 50 pixels for the score/level text at the top).

Multiplayer

You should create a server that listens for connections on a particular port, invoked as:

```
./server <port>
```

Then the client can connect to the server via:

```
./client <server name/IP> <port>
```

When choosing a port number, please pick some random number in the range 1024 – 65535, to avoid accidentally conflicting with another group on the same ilab server.

Each client connection should be handled on a separate thread in the server. The server should support at least 4 concurrent players. You can make the game either collaborative or competitive (as a design decision, not necessarily an option for the players). In either case, players that join the same server collect tomatoes on the same grid and move to the next level at the same time. Player movements should be visible to all other connected clients in real time.

If it's collaborative, the score is their collective total. If competitive, you should modify the UI to display the score for each player.

Once a game is in progress, a new client that connects to it should be informed of the current state, rather than starting at level 1, score 0. It's probably easiest to make the server store the definitive state of the game, and for the clients to mainly act as viewers of this data. The server-client protocol is up to you.

Be sure to use the appropriate synchronization mechanisms in the server when manipulating shared data.

Compiling and testing

Building on ilab

A Makefile is provided in `hw5.tgz`. To build the client on ilab, just run `make`. Since the ilab machines don't have the SDL2 development libraries installed, they're provided in the `inc` and `lib` directories. The Makefile informs gcc of these paths so it can compile. Note that the client requires a graphical display to run, so you should connect via x2go or similar.

To run the client, you need to set the path for finding dynamic libraries. Assuming you're using bash, you can prepend the hw5/lib directory to your `LD_LIBRARY_PATH` environment variable with this command (modifying the exact path as needed):

```
export LD_LIBRARY_PATH="$HOME/cs214/hw5/lib:$LD_LIBRARY_PATH"
```

Once that is set, you should be able to run the client directly with `./client`. Alternatively, the Makefile has a rule that will run the client with `LD_LIBRARY_PATH` set, so you can run the client with `make runclient`. If you see a message like this, you may have forgotten to set `LD_LIBRARY_PATH`:

```
./client: error while loading shared libraries: libSDL2_image-2.0.so.0:
          cannot open shared object file: No such file or directory
```

Building locally

If you want to build on a local Linux install, you'll need the required SDL libraries:

```
sudo apt install libsdl2-dev libsdl2-image-dev libsdl2-ttf-dev
```

Then, build with `make -f Makefile.local`.

Other OSs

SDL does support MacOS and Windows, but we only officially support Linux for this project.

Submission

Please be sure to test your code on ilab before submitting. Please submit the assignment on Canvas as a tar file `hw5.tar` that, when expanded, produces a `hw5` directory (possibly with additional .c/.h files):

```
hw5
├── Makefile
├── client.c
├── server.c
├── inc
├── lib
└── resources
```

If you work in a group, please include a `README.txt` file with the names and netIDs of everyone in your group. Only one person in the group should submit on Canvas.