

Lecture 15

Final Project Overview

CS211 – Fundamentals of Computer Programming II
Branden Ghena – Winter 2022

Slides adapted from:
Jesse Tov

Administrivia

- Homework 6 due Thursday
 - Remember that this is the last one!
 - Also, slip days aren't applicable to final project

- Final project starting!
 - Proposals are due Friday
 - More details right now!

Today's Goals

- Explain the what, why, and how of final projects
- Explore GE211 functionality not used in the homeworks
- Demonstrate some additional games you'll get as sample code
- Practice the creation of a GE211 game

Getting the code for today

- Download code in a zip files from here:
https://nu-cs211.github.io/cs211-files/hw/project_demos.zip
https://nu-cs211.github.io/cs211-files/lec/15_finalProject.zip
- Extract code wherever
- Open with CLion
 - Make sure you open the folder with the CMakeLists.txt

Outline

- **Final Project Overview**
- Additional GE211 Functionality
- Demo Games
- Game Motion Planning

Goals of the Final Project

- Focus on something that interests you
 - Pick anything you like (that's the right difficulty)
 - Chance to apply creativity and make something fun
- Program without safety rails or constraints
 - Starter code is very minimal
 - No specification with required functions to implement
 - You get to design how the code works
 - You can base your design off examples though!

Timeline

- https://nu-cs211.github.io/cs211-files/hw/final_project.pdf
- Friday, February 25 - Proposal
 - This week! (but only requires a one-sentence proposal)
- Tuesday, March 01 - Specifications
 - Next week
- Friday, March 11 - Code due
 - Last Friday of classes
 - 2 weeks after homework 6 is due!
- Sunday, March 13 - Evaluation guide
 - So we can grade them during exam week

Making proposals

- Something that interests you
 - Games are most common
 - I'll let you know if it's too easy or too complicated
- Good sources of inspiration
 - Classic arcade games
 - 2D mobile games
 - Board games
- Common problematic submissions
 - Pong, Snake game, Space Invaders, Flappy Bird
 - Any of the demo games: Keyracer, Bejewled, Asteroids

Making specifications

- List of 10-12 functionalities that your project will have
 - This is where difficulty is *really* decided
 - Grade is determined by whether you meet the specifications you create
- This is an iterative process
 - Submit spec items
 - Hear back from shepherd about what's good and bad
 - Make updates and repeat
- Goal:
 - Difficult enough to help you learn
 - Easy enough to complete

How to get started

1. Start with the model
 - Make the simplest version of the game that can do `_anything_`
2. Then implement a View and Controller so you can play it
 - Again, focus on the simple parts first
3. Then go back to model and add features
4. Finally, go back to View and Controller and add features
 - Sound, Better Graphics, etc.

Remember that simpler is often better

- If you're making a board game, you could take all of `board.cxx` and `board.hxx` and reuse it in your project
 - But it's complicated and you'll likely have to adjust some things for your game
 - Likely not the simplest path
- Alternative options
 - `std::vector<Posn>` track board locations for each player
 - `std::unordered_map<Posn, Player>` mark player for each location

Outline

- Final Project Overview
- **Additional GE211 Functionality**
- Demo Games
- Game Motion Planning

GE211 you've already used

- Abstract game class
 - `draw()`, `on_frame()`
- Events
 - Mouse and keys
 - Includes keyboard keys such as shift, ctrl, alt, and arrow keys
- Geometry
 - `Posn`, `Rect`, `Dim`
- Basic sprites
 - Rectangles and Circles of multiple colors

Additional GE211 Features

- **Resources files**
- Audio
- Advanced Sprites
- Sprite Manipulations
- Timer

Resources files

- Add a Resources/ directory to the project root
 - next to src/ and test/
- Put files into it that you want your game to access while running
 - Configurations
 - Level layouts
 - Images
 - Audio files

Accessing Resource files

- `ge211::open_resource_file(std::string const& filename)`
 - <https://tov.github.io/ge211/namespacege211.html#a2dadd7cd96f1642d432e9d63de63f00c>
 - Finds the filename specified and opens it for you
 - Don't specify Resources/, just the filename
 - Returns an `std::ifstream`
- Access the data within the `std::ifstream` with `>>`
 - Just like `stdin`
- Submitting Resources files:
 - Autograder puts everything that's not `*.cxx` or `*.hxx` into Resources/
 - Note: `test*.cxx` and `*test.cxx` go into `test/`

Additional GE211 Features

- Resources files
- **Audio**
- Advanced Sprites
- Sprite Manipulations
- Timer

Audio in GE211

- One Mixer controls all sounds for the game
 - https://tov.github.io/ge211/classge211_1_1audio_1_1_mixer.html
- Can continuously play one Music_track (background music)
 - https://tov.github.io/ge211/classge211_1_1audio_1_1_music_track.html
 - play, pause, resume, rewind, set_volume
- Can play short Sound_effects
 - https://tov.github.io/ge211/classge211_1_1audio_1_1_sound_effect.html
 - play, pause_all, resume_all
 - Can support several sound effects at once
 - Hardware dependent

Using audio

- How to get access to the mixer
 - Call `mixer()` inside the Controller
 - (Actually inside whatever inherits from `Abstract_game`)
- How to get a `Music_track` or `Sound_effect`
 - Call constructor with a filename string
 - Name of a file in Resources/
 - WAV, MP3, FLAC, MID, ABC, OGG, etc.
- Various sound effects and music can be found online

Additional GE211 Features

- Resources files
- Audio
- **Advanced Sprites**
 - **Text Sprites**
 - **Image Sprites**
- Sprite Manipulations
- Timer

Text Sprites

- Creates a sprite out of a string of text
 - Text, Color, and Font are configurable through a Builder
 - Placed on screen in `draw()` just like any other sprite
 - A little bit of work to manipulate though
- Text sprite can be reconfigured as needed
 - https://tov.github.io/ge211/classge211_1_1sprites_1_1_text_sprite.html
 - First use a Builder to create the text
 - https://tov.github.io/ge211/classge211_1_1sprites_1_1_text_sprite_1_1_builder.html
 - Then call `reconfigure()` with the Builder as the argument

Text sprite example

- **Keep sprites and fonts as private members of View**

```
unsigned int score;  
ge211::Posn<int> score_position;  
ge211::Font sans18{"sans.ttf", 18};  
ge211::Text_sprite score_sprite;
```

- **In draw(), reconfigure the string as needed**

```
ge211::Text_sprite::Builder current_score(sans18);  
current_score << score;  
score_sprite.reconfigure(current_score);  
set.add_sprite(score_sprite, score_position);
```

Image Sprite

- `Image_sprite(std::string const& filename)`
 - Creates a sprite out of a given image
 - Uses the image's dimensions in pixels
Transparency in images works!
- Filename comes from Resources/

Additional GE211 Features

- Resources files
- Audio
- Advanced Sprites
- **Sprite Manipulations**
- Timer

Applying Transforms to sprites

- What if your image sprite is larger than you want?
- Or if you want to rotate a sprite
- Transforms!
 - https://tov.github.io/ge211/classge211_1_1geometry_1_1transform.html
 - Enable rotation, scaling, and flipping sprites
 - Passed in as an alternative final argument to draw()
 - https://tov.github.io/ge211/classge211_1_1sprite_set.html#ad20a59df594c869b26e222da98c6161d

Additional GE211 Features

- Resources files
- Audio
- Advanced Sprites
- Sprite Manipulations
- **Timer**

Timers allow durations to be tracked

- Create a Timer() and start it
 - Later check it and you can see how long it was running
 - Allows you to determine how long some player action took
- Timer class
 - https://tov.github.io/ge211/classge211_1_1time_1_1_timer.html
 - Returns a Duration
 - https://tov.github.io/ge211/classge211_1_1time_1_1_duration.html
 - Which you can request time from in seconds or milliseconds

Break + Question

- There's an easier way to track time and perform actions after a certain amount of time has passed
- How would we use `on_frame(double dt)` to do so?

Break + Question

- There's an easier way to track time and perform actions after a certain amount of time has passed
- How would we use `on_frame(double dt)` to do so?
 - `dt` is in units of seconds
 - Usually $1/60^{\text{th}}$ of a second
 - Keep a local variable that you add `dt` to each time `on_frame()` is called
 - Reset the variable to zero whenever you need to start counting
 - If variable is greater than some amount, trigger action

Outline

- Final Project Overview
- Additional GE211 Functionality
- **Demo Games**
- Game Motion Planning

A note on these demos

- All kinds of complicated C++ stuff going on here
 - Some of it is good
 - Some of it is just messy
- Purpose of the demo code
 1. Inspire you about what is possible
 2. Provide useful reference code
- Not recommended to use one of these as a starting point
 - Too much stuff going on that wouldn't be relevant

Keyracer

- Practice typing words under time pressure
- Loads information from a Resource file containing all English words
 - `load_dictionary()` in `controller.cxx`
 - As you'll see, this dictionary is a bit dubious...
- Timer bar counts down until you've "missed" the letter
 - Also miss if you hit the wrong key
 - Counts down time in `on_frame()`
 - Uses a Transform to scale the timer bar

Bejeweled

- Align groups of colored circles in a grid to score them
 - Makes the group disappear, scoring points
 - More colored circles fall down from the top
- Uses background music (optionally) and sound effects
 - Sound effects play when scoring or when an invalid move is made
- “Animates” steps when scoring
 - Circles disappear from the screen over several frames
 - Then circles fall down from top over several frames

Asteroids

- Avoid or shoot asteroids in a spaceship that has momentum
 - Asteroids that are shot break into multiple smaller pieces
 - Ship gains or loses velocity as you hold arrow keys
- Uses image sprites for objects in the game
- Objects rotate in addition to moving
 - `on_frame()` updates position, velocity, rotation, and angular velocity
 - `draw()` applies Transforms to objects
 - Place at position, Rotate to rotation, Scale based on mass
- Tracks key down/up to start and stop actions

Getting demo code

- https://nu-cs211.github.io/cs211-files/hw/project_demos.zip
- Includes three separate projects
 - Keyracer
 - Bejeweled
 - Asteroids

Break + Request

- Let me know if you want a demo of anything specific
 - Either after class or on Campuswire
- We have time to build a few game examples together in class
 - Today we're going to demonstrate:
 - Image sprites
 - Moving sprites around the screen
 - Text sprites
 - We'll continue these in the coming lectures

Outline

- Final Project Overview
- Additional GE211 Functionality
- Demo Games
- **Game Motion Planning**

Multi-lecture project example

- Starting from https://nu-cs211.github.io/cs211-files/hw/final_project.zip
- We'll add features as we go
 - This will *definitely* not be finished today

Plan for game

- Image sprite that represents a character in the game
 - Moves towards a given position at a set velocity
- Text sprite to explain what position is being moved to
- Each character keeps a list of positions to move to
 - Moves towards the first position until it reaches it
 - Then starts moving towards the next position
- Add to list of positions with mouse clicks

Initial Character class

- Data members
 - Image_sprite sprite_
 - Posn<float> position_
- Interface
 - Constructor (from string for filename)
 - Getters/Setters for data members

Drawing the sprite

- Add sprite image to Resources/
- Add character to Model as a private member
 - Probably a `std::vector` of characters
- Add getter to allow View to access characters vector
- Update View to iterate through the characters and draw each one

Add motion to Character class

- Data members
 - Image_sprite sprite_
 - Posn<float> position_
 - float velocity_
 - Posn<float> destination_
- Interface
 - Constructor (from string for filename)
 - Getters/Setters for data members
 - update(double dt) called from on_frame()
 - distance_to_position_() helper function

Making the sprites move

- Add initial destinations upon creation in the Model
- Add `on_frame()` function to Controller and Model
 - Call Model's `on_frame()`
 - Then call each character's `on_frame()`

Add a text sprite to explain each character's movement

- View gets new private members
 - `ge211::Text_sprite explanation_`
 - `ge211::Font sans28_`
- Build output string in `draw()`
 - Create an `Image_sprite::Builder`
 - Set a font and a Color
 - Set the string to be displayed based on the character
 - Reconfigure the `Image_sprite`
 - Add the sprite so it appears

Upgrade characters to hold a list of destinations

- Probably want to use an `std::queue`
 - `push()` positions to the end of the queue
 - `pop()` positions from the front of the queue
- Change to the next destination after we reach it
 - Occurs in `on_frame()`
- Make sure the initial destination is the initial position
 - Or we'll start moving somewhere right away

Use mouse clicks to specify waypoints for a character

- Respond to mouse clicks in the Controller
 - Forward click to the model to act upon
- Model uses mouse click to add destination for first character

Outline

- Final Project Overview
- Additional GE211 Functionality
- Demo Games
- Game Motion Planning