Catchy Fish

CIS 25 Final

Catchy Fish

Catchy Fish is a text based fishing minigame using the Ncurses library to draw on the screen.

Challenges I faced

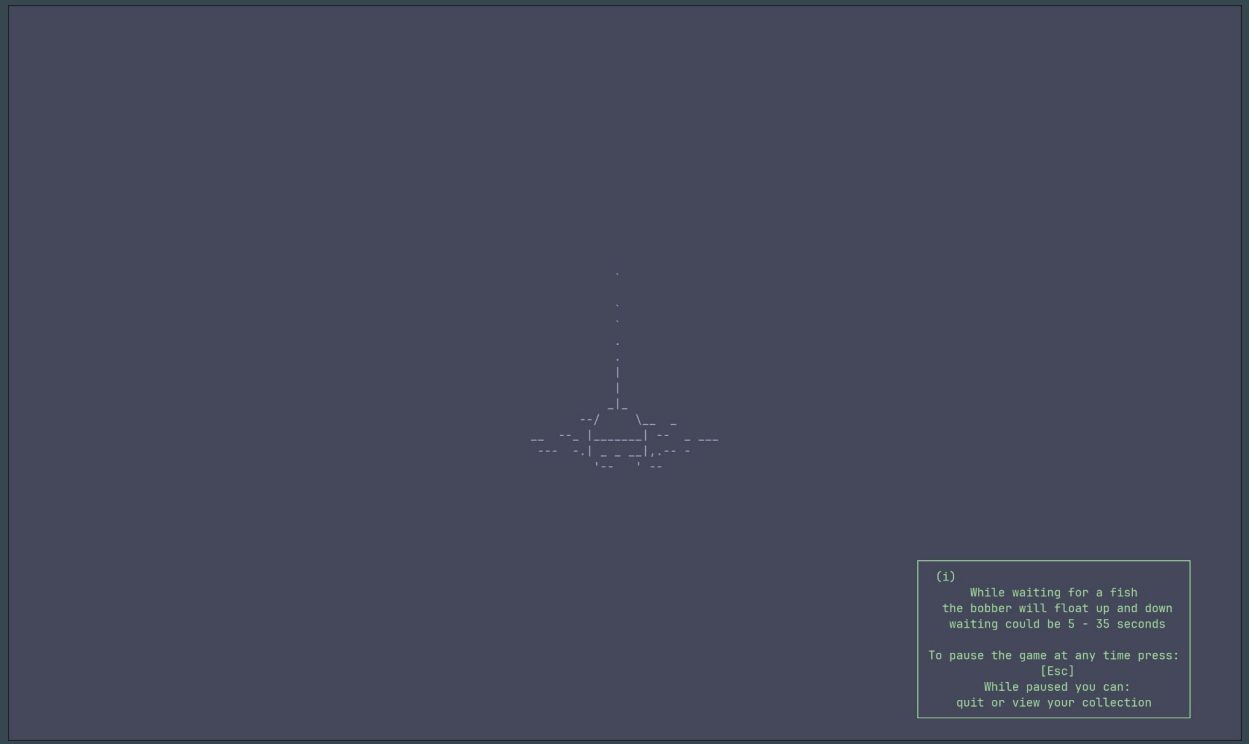
One challenges I faced was organizing my code as this was a bigger project then I have made in the past.

Another challenge I faced was learning how to use the Ncurses library.

GUI

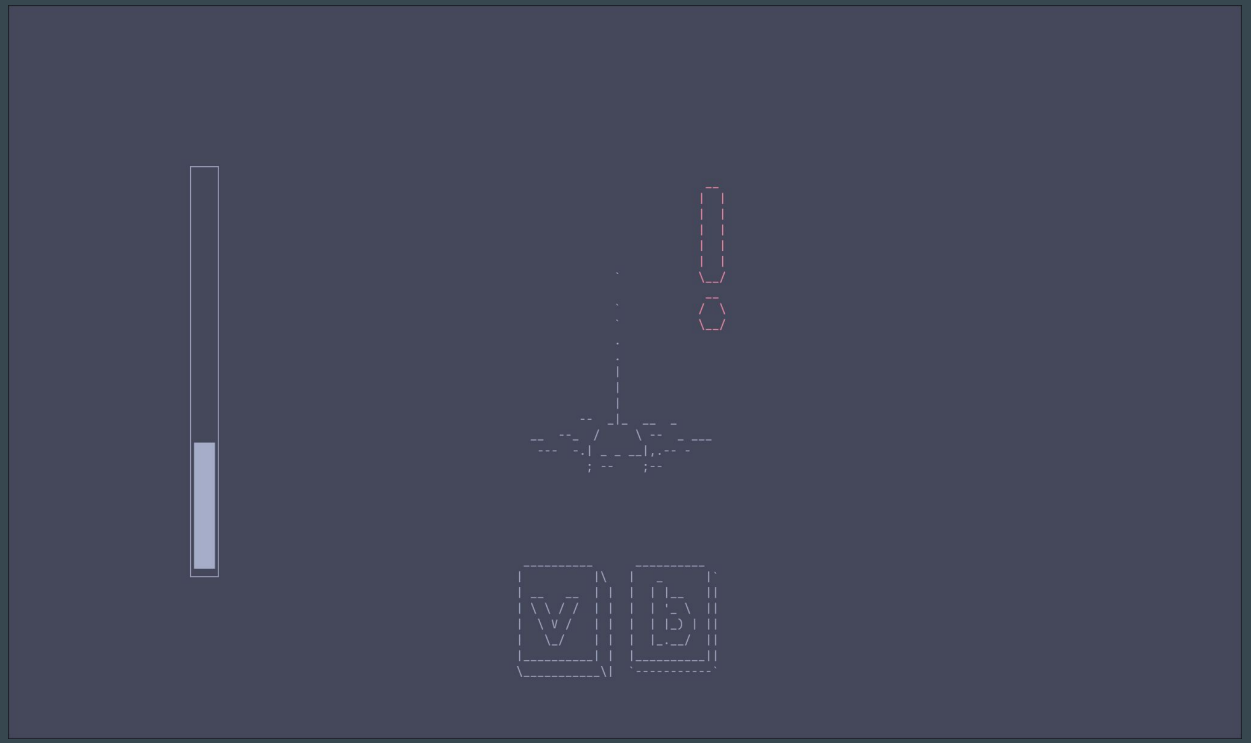
Main Menu

- To draw on the terminal I use the C library, Ncurses.
- The Menu has 3 buttons
  - Play - start game
  - Tutorial - start Tutorial
  - Quit - close game
- It also has an info box, informing new users how to navigate menus.



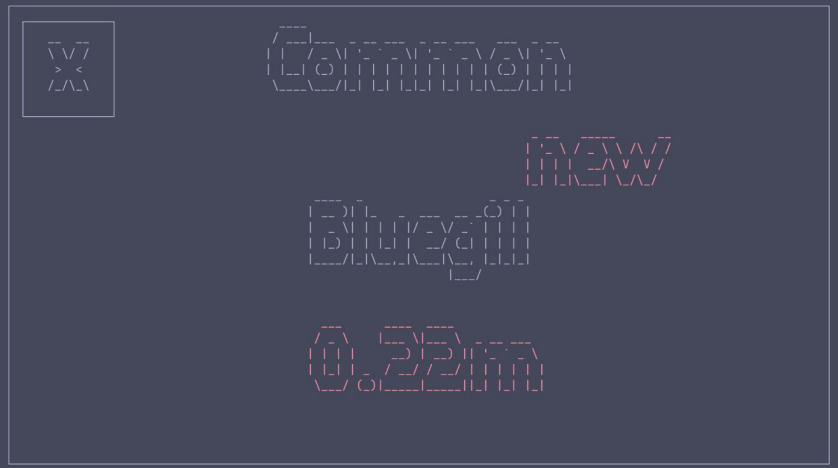
Waiting

- While waiting to catch a fish the user will see this screen
- In the middle you will see a bobber bob up and down every once in a while
- If the user chose Tutorial they will see the text in the bottom



Catching

- When a fish is on the hook you will see this screen
- As you press [v] or [b] the user can see the progress bar increase



Caught

- Once the fish is caught the user will see this screen
- The word “new” will be shown when it is a new fish caught
- The number will be red if it is bigger than a previously caught fish
- The color of the rarity text will also change depending of the rarity

Paused

- When the user presses [Esc] the game will pause and the user will see this menu
- The Menu has 3 buttons
  - Resume - start game
  - Collection - show fish collection
  - Quit - close game

Collection

- When the user chooses to view there collection they will see this screen
- The Menu can switch between showing the different rarities of fish
- If the user has not caught a specific fish it will appear as “???”

```
181   int ch;
182   bool loop = true;
183   while (loop) {
184
185     switch (program_state.current_state) {
186     case MainMenu:
187       ch = getch(); // read key pressed
188       // if menu changed
189       if (main_menu.handle_input(ch, program_state)) {
190           main_menu.draw();
191       }
192
193       // if state has changed
194       if (program_state.current_state != MainMenu) {
195           mousemask(0, NULL); // disable mouse reading
196           main_menu.clear();
197
198           if (program_state.current_state == Waiting || program_state.current_state == TutorialWaiting) {
199             if (program_state.current_state == TutorialWaiting)
200               // tutorial has a set fish encounter time
201               fish_encounter_time = time(NULL) + 15;
202             else
203               fish_encounter_time = time(NULL) + chosen_fish.get_fish_delay() +
204                                     (int)(rand() % chosen_fish.get_random_fish_delay());
205             halfdelay(1); // make getch() wait 1/10sec if nothing happens return ERR
206           }
207       }
208       break;
```

<- If menu has changed redraw it
// update and draw

<- If state has changed prepare for the next state

Control Flow

- The program uses a state machine to control what is happening
- Most states will follow a similar pattern to the Main menu

```
451     case Paused:
452       ch = getch();
453       if (pause_menu->handle_input(ch, program_state)) {
454           pause_menu->draw();
455       }
456       // if user clicked collection button
457       if (program_state.current_state == ViewCollection) {
458         pause_menu->clear();
459         pause_menu = &collection_menu;
460         pause_menu->draw();
461         program_state.current_state = PausedViewCollection;
462       }
463
464       if (ch == ESC) {
465         program_state.current_state = PreviousState;
466       }
467
468       if (program_state.current_state != Paused && program_state.current_state != PausedViewCollection) {
469           if (program_state.current_state == PreviousState) {
470             program_state.current_state = program_state.previous_state;
471
472             if (program_state.current_state == TutorialWaiting || program_state.current_state == Waiting ||
473                 program_state.current_state == TutorialCatching || program_state.current_state == Catching) {
474               if (fish_encounter_time != -1)
475                 fish_encounter_time += time(NULL) - pause_menu_timer;
476               halfdelay(1);
477             }
478           }
479           pause_menu->clear();      //reset pause menu
480           pause_menu = &escape_menu;//
481           escape_menu.reset();      //
482           collection_menu.reset();  //
483           if (program_state.current_state != Caught)
484             mousemask(0, NULL);
485           program_state.previous_state = Paused;
486       }
487       break;
```

<- If menu has changed redraw it
// update and draw

If state has changed prepare for the next state

Second Example

More In Depth

- Like the main menu it handles input and draws pause menu if something happened.
- If the state has changed to ViewCollection draw the collection
- If the player presses [Esc] change state to previousState
- Final if state is not paused anymore prepare for the next state by clearing the current one and setting up curtain requirements the next state needs

Classes

For this project I had created multiple header files.

- window.cpp
- bar.cpp
- menu.cpp
- textBox.cpp
- fish.cpp
- collection.cpp
- globalState.h
- textAssets.h

window.h

```
#pragma once

class Window {
public:
  Window(int height, int width, int start_y, int start_x, bool is_boxed = true);

  virtual void clear() const;
  virtual void draw() const = 0;

  void set_is_boxed(bool is_boxed);
  bool get_is_boxed() const;

  static void box(int height, int width, int start_y, int start_x, int color = 0);

protected:
  int height;
  int width;
  int start_y;
  int start_x;

  void box(int color = 0) const;

private:
  bool is_boxed;

};
```

The window class was made to create easier use with Ncurses by creating a structure of inherited classes and provide helpful variables and functions

- start_y and start_x
- width and height
- is_boxed
  - Is window have a box
- box()
  - Draw a box based around window area

```cpp
#pragma once

#include <ncurses.h>

#include "window.h"

class ProgressBar: public Window {
public:
  ProgressBar(float progress, int height, int width, int start_y, int start_x);
  ProgressBar(int height, int width, int start_y, int start_x);

  float get_progress() const;
  void set_progress(float progress);

protected:
  float progress;

  int empty_char;
  int full_char;
};

class VerticleProgressBar: public ProgressBar {
public:
  VerticleProgressBar(float progress, int height, int width, int start_y, int start_x);
  VerticleProgressBar(int height, int width, int start_y, int start_x);

  void draw() const override;
};

class HorizontalProgressBar: public ProgressBar {
public:
  HorizontalProgressBar(float progress, int height, int width, int start_y, int start_x);
  HorizontalProgressBar(int height, int width, int start_y, int start_x);

  void draw() const override;
};
```

bar.h

As the name implies it the ProgressBar class is a progress bar. It will draw a progress bar based on the progress variable (0.0-1.0).

There are two inherited classes, Vertical/HorizontalProgressBar, each with their own implemented draw functions.

menu.h

```cpp
#pragma once

#include <ncurses.h>
#include <string>
#include <vector>

#include "globalState.h"
#include "textBox.h"

class MenuButton : public Window {
public:
  MenuButton(int height, int width, int start_y, int start_x, state button_action, char key = 0);

  int get_key() const;
  bool get_is_selected() const;
  void set_is_selected(bool is_selected);

  void draw() const override;
  state get_action() const;

  bool is_mouse_on_button(MEVENT& event) const;

private:
  char key;
  bool is_selected;
  state button_action;
};
```

Menu.h defines three classes all relating to interactive menus. The first class is MenuButton.

The class stores the state the game should change to if it is pressed. It also has a helper function for checking if mouse in on the button.

```
29 class Menu : public Window {
30 public:
31   Menu(int height, int width, int start_y, int start_x, std::vector<MenuButton> menu_buttons, std::vector<TextBox*> menu_texts);
32   ~Menu();
33
34   virtual void draw() const override;
35   virtual bool handle_input(int input_key, GameState& game_state);
36
37   virtual void reset();
38
39 private:
40   int selected_button_index;
41   std::vector<MenuButton> menu_buttons;
42   std::vector<TextBox*> menu_texts;
43 };
44
45 class MenuCollection : public Menu {
46 public:
47   MenuCollection(std::vector<Menu*> menus);
48
49   int get_selected_menu_index() const;
50
51   void draw() const override;
52   bool handle_input(int input_key, GameState& game_state) override;
53
54   void reset() override;
55
56 private:
57   std::vector<Menu*> menus;
58   int selected_menu_index;
59 };
```

menu.h

The two other classes are Menu and MenuCollection.

Menu handles the users inputs to control the Menu.

MenuCollection allows for multiple menus.

textBox.h

```cpp
#pragma once

#include <vector>
#include <string>

#include "window.h"

class TextBox : public Window {
public:
  TextBox(std::vector<std::string> text, int height, int width, int start_y, int start_x, bool is_boxed, int color = 0);
  TextBox(std::vector<std::string> text, int height, int width, int start_y, int start_x);

  static void draw(const std::vector<std::string>& text, int height, int width, int start_y, int start_x, bool is_boxed, int color);
  static void draw(const std::vector<std::string>& text, int height, int width, int start_y, int start_x, int color = 0);
  static void draw(const std::vector<std::string>& text, int start_y, int start_x, int color = 0);

  virtual void draw() const;

protected:
  std::vector<std::string> text;
  int color;
};

class TextBoxCentered: public TextBox {
public:
  TextBoxCentered(std::vector<std::string> text, int height, int width, int start_y, int start_x, bool is_boxed, int color = 0);
  TextBoxCentered(std::vector<std::string> text, int height, int width, int start_y, int start_x);

  static void draw(const std::vector<std::string>& text, int height, int width, int start_y, int start_x, bool is_boxed, int color);
  static void draw(const std::vector<std::string>& text, int height, int width, int start_y, int start_x, int color = 0);

  void draw() const override;
};
```

The TextBox and TextBoxCentered classes provide an easy way to draw multiple lines of text to the screen.

The TextBoxCentered class will draw the text centered to its area.

```cpp
#pragma once

#include <vector>
#include <string>

enum Rarity {
  Common,
  Uncommon,
  Rare,
  Legendary,
};

enum FishVariety {
  Catfish,
  Bass,
  Cod,
  Trout,
  Bluegill,
  Salmon,
  Crawfish,
  Eel,
  Octopus,
};

class Fish {
public:
  Fish(FishVariety name, float fishing_power, float fish_strength,
       int fish_delay, int random_fish_delay, Rarity rarity, float min_size,
       float max_size);

  Fish(FishVariety name);
  Fish(FishVariety name, float size);

  FishVariety get_variety() const;
  float get_fishing_power() const;
  float get_fish_strength() const;
  int get_fish_delay() const;
  int get_random_fish_delay() const;
  Rarity get_rarity() const;
  float get_min_size() const;
  float get_max_size() const;
  float get_size() const;
```

```cpp
  // return the variety as a string
  std::string get_name() const;

  static int get_rarity_color(Rarity rarity);
  static const std::vector<std::string>& get_variety_text(FishVariety name);
  static const std::vector<std::string>& get_rarity_text(Rarity rarity);

private:
  void init_rarity(Rarity rarity);
  void init_size(float min_size, float max_size);

  FishVariety name;

  // amount bar increases on click
  float fishing_power;
  // amount bar decrease per 1/10 second
  float fish_strength;

  // the delay for waiting to catch fish
  // timer = time(NULL) + fish_delay + (int)(rand() % random_fish_delay)
  int fish_delay;
  int random_fish_delay;

  // number of fish added to the selection pool
  Rarity rarity;

  // bounds of fishes size
  float min_size;
  float max_size;

  float size;
};
```

fish.h

Stores fish information

collection.h

```h
#pragma once

#include <vector>

#include "fish.h"

class Collection {
public:
  Collection();

  void load_from_file();
  void save_to_file();

  void add(Fish fish);
  bool is_fish_inside(FishVariety fish);
  bool is_fish_bigger(Fish fish);

  float get_fish_size(FishVariety fish_variety);

private:
  std::vector<Fish> collection;
};
```

The Collection class stores the fish the user has caught. It provides helpful functions for comparing fish and loads and saves to file.

```
Bluegill,0.22
Trout,0.55
Salmon,0.5
Bass,0.46
Eel,2.64
Crawfish,0.15
```

Example of file saving. It is stored in the file collection.csv
{fish}, {size}

```
#pragma once

enum state {
  MainMenu,
  TutorialWaiting,
  Waiting,
  TutorialCatching,
  Catching,
  Caught,
  Paused,
  ViewCollection,
  PausedViewCollection,
  Quit,
  PreviousState,
  NextMenu,
  PreviousMenu,
};

struct GameState {
  enum state current_state;
  enum state previous_state;
};

```

globalState.h

The globalState.h file defines the games possible states and a GameState struct.

The GameState struct is used to define the current state and for menus to go back to a previous state.

```cpp
#pragma once

#include <vector>
#include <string>

namespace Assets {

  const std::vector<std::string> quit_text = {
    "  ___        _ _    ",
    " / _ \\ _   _(_) |_ ",
    "| | | | | | | | __|",
    "| |_| | |_| | | |_ ",
    " \\__\\_\\\\__,_|_|\\__|",
  };

  const std::vector<std::string> play_text = {
    " ____  _             ",
    "|  _ \\| | __ _ _   _ ",
    "| |_) | |/ _` | | | |",
    "|  __/| | (_| | |_| |",
    "|_|   |_|\\__,_|\\__, |",
    "               |___/",
  };

  const std::vector<std::string> title_text = {
  "  ____       _            _____ _      _          ",
  " / ___|__ _| |_ ___| |__  _   _  |  ___(_)___| |__   ",
  "| |   / _` | __/ __| '_ \\| | | | | |_  | / __| '_ \\ ",
  "| |__| (_| | || (__| | | | |_| | |  _| | \\__ \\ | | |",
  " \\____\\__,_|\\__\\___|_| |_|\\__, | |_|   |_|___/_| |_|",
  "                          |___/                     ",
  };

  const std::vector<std::string> title_text2x = {
  "    ________                __                __                                 __________  __          __          ",
  "   /        |              |  |              |  |                               |          |/  \\        |  |         ",
  "  /   ______| ____   __ |  | __  ______ |  | ____    __        __      |   _______|\\__/ ______ |  | ____    ",
  " |   |       /    \\|   |      |/      ||  ''    \\  |  |    |  |    |  |         |  |/       |  ''    \\  ",
  " |   |      /   __    |  ____/   ____||    __    \\ |  |    |  |    |  | __      |  /   ____ |    __    \\ ",
  " |   |     |   /  |   |  |  |   /       |   |  |  | |  |    |  |    |     |    | \\\\       \\\\|   |  |   |",
  " |   |____ |   \\__|    |  |__|   \\____ |   |  |  | |  | __ |  |    |    __|    |  |\\\\____   \\\\   |  |   |",
  "  \\\\         \\\\       ,  |\\\\    \\\\        ||   |  |  | \\\\          |    |   |       |  ||       /   |  |   |",
  "   \\\\ _______|\\\\ ____/|__| \\\\____\\\\ ______||__ |  |__|  \\\\______,  |    |___|       |__||______/|___|  |___|",
  "                                                    ,----'   /                                                  ",
  };

}
```

textAssets.h

This file is used to store the text art used throughout the game. The assets are stored is a Assets namespace. The TextBox class can be used to display these on the screen.