ual: creative computing institute

C++ Day 3

Introduce Object-Oriented Programming

2023

Jasper Zheng

Object-Oriented Programming

Using OOP, we can create our own data type by combining related variables and functions into a unit.

For example, if we need to keep track of the statistics of all characters in our game:

Each character will need a name, health (HP), attack damage (AD), attack resistance (AR), and a list of items carried by the character.

We can create a class called "character", which maintains all these statistics of a single character.

Object-Oriented Programming

Class

We combine related variables and functions into a unit to describe a real-world concept. We call this unit a class.

"Character" is a class, "Player" is a class.

Object

We can create many instances of a class, each of these instances is called an object.

A character called Bob is an object (a character instance), a different character called Alice is also an object (another character instance).

```
class Character {
public:

    string name;
    long int hp, ad, ar;
    vector<Item> items;

    Character(string name, int hp, int ad, int ar);
    void respawn();
};


int main() {
    Character bob ("bob", 2000, 400, 100);
    Character alice("alice", 1000, 300, 200);

    bob.respawn();
    alice.respawn();

    return 0;
}
```

Object-Oriented Programming

Property

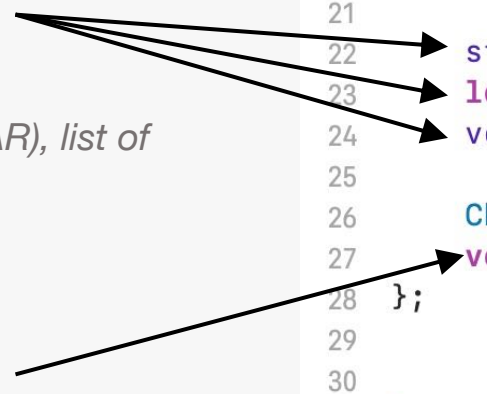
Properties are variables defined in a class.

name, health (HP), attack_damage (AD), attack_resistance (AR), list of items are properties of a character.

Method

Methods are functions defined in a class.

`Respawn()` is a method of a character. All characters can respawn, but they may respawn with different HP, depending on their health.



```cpp
class Character {
public:

    string name;
    long int hp, ad, ar;
    vector<Item> items;

    Character(string name, int hp, int ad, int ar);
    void respawn();
};


int main() {
    Character bob ("bob", 2000, 400, 100);
    Character alice("alice", 1000, 300, 200);

    bob.respawn();
    alice.respawn();

    return 0;
}
```

Object-Oriented Programming

Constructor

A constructor is a function that is automatically called when an object of a class is created. It helps you to define the initial properties of an object.

The constructor has been called here.

```
19  class Character {
20  public:
21
22      string name;
23      long int hp, ad, ar;
24      vector<Item> items;
25
26      Character(string name, int hp, int ad, int ar);
27      void respawn();
28  };
29
30
31  int main() {
32      Character bob ("bob", 2000, 400, 100);
33      Character alice("alice", 1000, 300, 200);
34
35      bob.respawn();
36      alice.respawn();
37
38      return 0;
39  }
```

Object-Oriented Programming

Constructor

A constructor is a function that is automatically called when an object of a class is created. It helps you to define the initial properties of an object.

The constructor function:

```
Character::Character(string name, int hp, int ad, int ar){
    // some code for initialisation..
    this->name = name;
    this->hp = hp;
    this->ad = ad;
    this->ar = ar;
}
```

```
class Character {
public:

    string name;
    long int hp, ad, ar;
    vector<Item> items;

    Character(string name, int hp, int ad, int ar);
    void respawn();
};


int main() {
    Character bob ("bob", 2000, 400, 100);
    Character alice("alice", 1000, 300, 200);

    bob.respawn();
    alice.respawn();

    return 0;
}
```

Arrange Your Code...

Headers

It's a good practice to list all properties and methods ahead of the `main()` function (like a placeholder), and later fill them in with actual codes.

The actual codes for the functions are implemented here.

```
19  class Character {
20  private:
21      long int hp, ad, ar;
22      vector<Item> items;
23  public:
24      string name;
25      Character(string name, int hp, int ad, int ar);
26      void respawn();
27  };
28
29  int main() {
30      Character bob("bob", 2000, 400, 100);
31      Character alice("alice", 1000, 300, 200);
32      bob.respawn();
33      alice.respawn();
34      return 0;
35  }
36
37
38  Character::Character(string name, int hp, int ad, int ar){
39      // some code for initialisation..
40      this->name = name;
41      this->hp = hp;
42      this->ad = ad;
43      this->ar = ar;
44  }
45  void Character::respawn() {
46      // some code for respawning...
47      cout << name << " respawned with hp: " << hp << endl;
48  }
```

Object-Oriented Programming

Public / Private

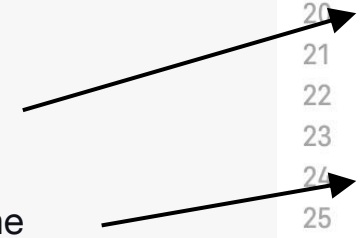
Private properties and methods are cannot be accessed (or viewed) from outside the class.

Public properties and methods are accessible from outside the class.

```
bob.name;   // “bob”

bob.hp;     // error
```

[Public / Private]



```
19  class Character {
20  private:
21      long int hp, ad, ar;
22      vector<Item> items;
23
24  public:
25      string name;
26      Character(string name, int hp, int ad, int ar);
27      void respawn();
28  };
29
30
31  int main() {
32      Character bob("bob", 2000, 400, 100);
33      Character alice("alice", 1000, 300, 200);
34
35      bob.respawn();
36      alice.respawn();
37
38      return 0;
39  }
```

Daily Code Jumpstart Choreography

	Mon	Tues	Wed	Thurs	Fri
9am-10am	---	Coaching aims	Daily Aims and Objectives	Daily Aims and Objectives	---
10am-13oo	---	Self-study Time	Self-study Time	Self-study Time	---
Break	---	Social Lunch	Social Lunch	Social Lunch	---
14oo-16oo	---	Self-study Time	Self-study Time	Self-study Time	---
18oo-19oo	---	QandA with Coach	QandA with Coach	QandA with Coach	---

Day 3 Tasks

- Task 1 – Two Videos on Object-Oriented Programming (approx. 40 mins)
- Task 2 – Upgrade the Tic-tac-toe (approx. 50 mins)

Task 1

1. Object-Oriented Programming

Task 1.1 - Introduction to OOP (<10 mins)

Watch this Introduction to OOP video about the four main principles in OOP

Task 1.2 - OOP in Practice (~30 mins)

Option 1:

- Read and follow this chapter on C++ Classes and Objects
- Read and follow this chapter on C++ Class Methods
- Read and follow this chapter on C++ Constructors

Option 2:

- Watch and follow this Practical tutorial on OOP in C++ (only 3:13:27 to 3:41:42 [Chapter Classes & Objects, Constructor Functions, and Object Functions])

Task 2

2. Better Tic-Tac-Toe

Recap:

Yesterday we made our basic tic-tac-toe game:

- we used a 2D array to represent the grid
- we also designed a program loop to model the interaction

```
int grid[y_dim][x_dim]; // grid: 0 for empty; 1 for player x; 2 for player o;
int marker; // marker: 1 for player x, 2 for player o;

void initialiseGrid();
void showGrid();
bool checkInput(int x, int y);

int main() {

    int x,y;
    initialiseGrid();
    showGrid();

    for (int i = 0; i < x_dim * y_dim; i++){

        marker = i % 2 + 1;

        while (true) {
            if (marker == 1){
                cout << "Player x \n";
            } else {
                cout << "Player o \n";
            }
            cout << "enter row: ";
            cin >> y;
            cout << "enter column: ";
            cin >> x;

            if (checkInput(x, y)){
                break;
            }
        }
        grid[y-1][x-1] = marker;
        showGrid();
    }
}
```

Task 2

2. Better Tic-Tac-Toe

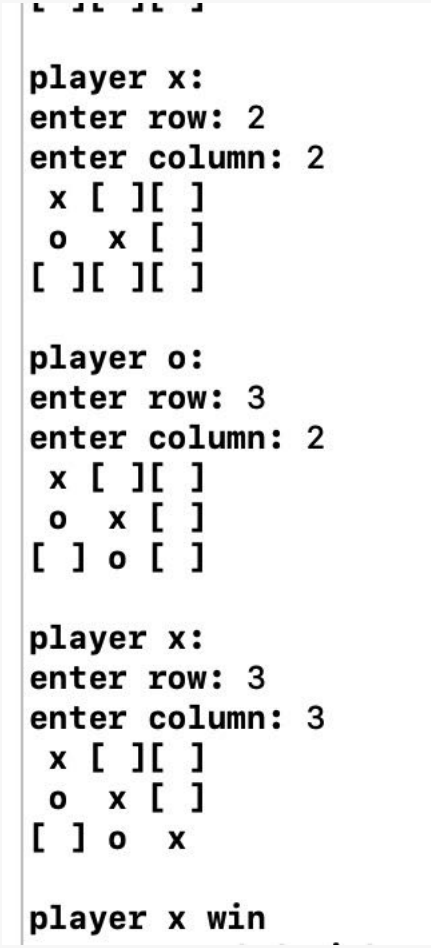
Task 2 - Better Tic-Tac-Toe (approx. 50 min):

Today we're going to upgrade our game to automatically check which player is the winner.

As our program grows, we may end up with a bunch of variables and functions all over the place.

Therefore, we're going to use OOP to re-organise our code.

More instructions on following slides ->



Task 2

2. Better Tic-Tac-Toe

Step 1 – Define System Components

We'll split the system in a Grid object and a Player object.

```
class Grid{};
```

The grid object handles everything happens on the grid, acting like a referee (i.e. add markers, initialise and print the grid, decide a winner, check if a cell is taken)

```
class Player{};
```

The player object represent a player, maintaining a player's index and markers, and handle inputs.

Task 2

2. Better Tic-Tac-Toe

```
class Grid{};
```

Properties:

```
int x_dim;
int y_dim;
vector<vector<int> >grid;
```

x_dim and y_dim represent the dimension of the grid, in a 3 x 3 grid they'll all be 3

Methods:

the constructor function that is going to run every time we define a new Grid object

```
Grid(int x_dim, int y_dim);
void initialiseGrid(int x_dim, int y_dim);
void showGrid();
bool isGameOver();
bool checkInput(int x, int y);
void placeMarker(int x, int y, int marker);

int checkRowCrossed();
int checkColumnCrossed();
int checkDiagonalCrossed();
```

checking every row, column and diagonal to see if there is a winner, will be called by the `isGameOver()` function (since the game will end if a winner shows up

Task 2

2. Better Tic-Tac-Toe

```
class Player{};
```

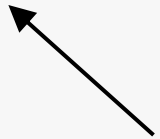
Properties:

```
int index;
char marker_char;
```

character “X” or “O”

Methods:

```
Player(int player_index);
void playersMove(Grid &grid);
```



there’s a “&” in front of the grid, it means we are passing a reference of the vector, instead of passing a copy of it, so changes made in this function is reflected in main()

Task 2

2. Better Tic-Tac-Toe

Step 2 – Check Winner

Right after a player makes a move, we check all rows, columns and diagonals.

When checking rows, we iterate through all rows and check if there's a row with three non-zero marks that match.

```
149 int Grid::checkRowCrossed(){
150     for (int i = 0; i < y_dim; i++){
151         int n = 0;
152         while (grid[i][n] == grid[i][n+1] and grid[i][n] != 0){
153             n += 1;
154             if (n == x_dim - 1){
155                 // If we find two pairs of matching marks in a row, return with the number at that place.
156                 return grid[i][n];
157             }
158         }
159     }
160     // If we checked all rows and found nothing, return 0
161     return 0;
162 }
```

We use similar approach to check columns in `int checkColumnCrossed();` and check diagonals in `int checkDiagonalCrossed();`

Task 2

2. Better Tic-Tac-Toe

Step 3 – Is Game Continue?

To decide whether the game is continuing after each move, we first run the three checking functions we defined in the last step, then check if there are empty spaces in the grid.

The game is not over if no winner shows up and there are still empty spaces.

```
118 bool Grid::isGameOver(){
119
120     // we are checking on every rows, columns and diagonals to see if there is a winner
121     // these three functions return 0 if there's no winner, return 1 or 2 indicating player 1 or 2 has win the game
122     int row = checkRowCrossed();
123     int col = checkColumnCrossed();
124     int dia = checkDiagonalCrossed();
125
126     if (row==1 or col==1 or dia==1){
127         cout << "player x win\n";
128         return true;
129     } else if (row==2 or col==2 or dia==2){
130         cout << "player o win\n";
131         return true;
132     }
133
134     // next, we check if the grid has no empty space, in that case the game ends with a draw
135     for (int y = 0; y < y_dim; y++){
136         for (int x = 0; x < x_dim; x++){
137             if (grid[y][x] == 0){
138                 // if we spot an empty space, return false so that the game is not over
139                 return false;
140             }
141         }
142     }
143     cout << "draw\n";
144     return true;
145 }
```

Task 2

2. Better Tic-Tac-Toe

Step 4 – Put Everything Together

It's a good practice to keep the main function minimal, and only contain high-level processes, so that our program is more maintainable.

Link to the full code

```
201 int main() {
202
203     Grid grid(3, 3);
204     Player player1(1);
205     Player player2(2);
206
207     int round = 0;
208     grid.showGrid();
209
210     while(!grid.isGameOver()){
211         if (round%2 == 0){
212             player1.playersMove(grid);
213         } else{
214             player2.playersMove(grid);
215         }
216         grid.showGrid();
217         round += 1;
218     }
219 }
```

Initialisation

Main game loop

Day 3 resources

CreativeApplications.Net: a community of art, media and technology

GitHub Quickstart

Missing Semester: a short course help you to gets familiar with terms like terminal, Linux command-line, git, version controls...

Map of Computer Science: a 10mins video explaining main subjects in computer science

Outlook: C++ at CCI

openFrameworks: a tool for creative coding (e.g. interactive moving images, generative arts / sounds, visualisation...)

Raspberry Pi: run openFrameworks in embedded systems for installations, synthesisers.

JUCE: a tool for making music app / plug-ins / virtual instruments

Unity: game engine

Unreal Engine: game engine

ual: creative computing institute

Concluding Week 3 Survey

https://artslondon.padlet.org/hbrueggemann/j2yr3zfwkap4v4rq

The password is **Jumpstart**.

Thank you, see you at CCI ☺

ual: creative computing institute

Thank you for joining ☺
Catch you at Welcome Week!

@ual_cci

@ual_cci

arts.ac.uk/cci