Spike: 10
Title: Sprites and Graphics

Author: Ryan Chessum, 102564760

Goals / deliverables:

- Code, see /19 – Spike – Sprites and Graphics /Messaging/
- Spike Report

Technologies, Tools, and Resources used:

List of information needed by someone trying to reproduce this work

- Visual Studio 2019
- C plus plus reference (https://www.cplusplus.com/reference/)
- Zorkish Specifications

Tasks undertaken:

- Download and install Visual Studio
- Create a new C++ project
- Plan out a design for a messaging system
- Import necessary classes
- Create a message class
- Create a blackboard class
- Add update function to entities
- Add code to entities so that they can send/receive messages

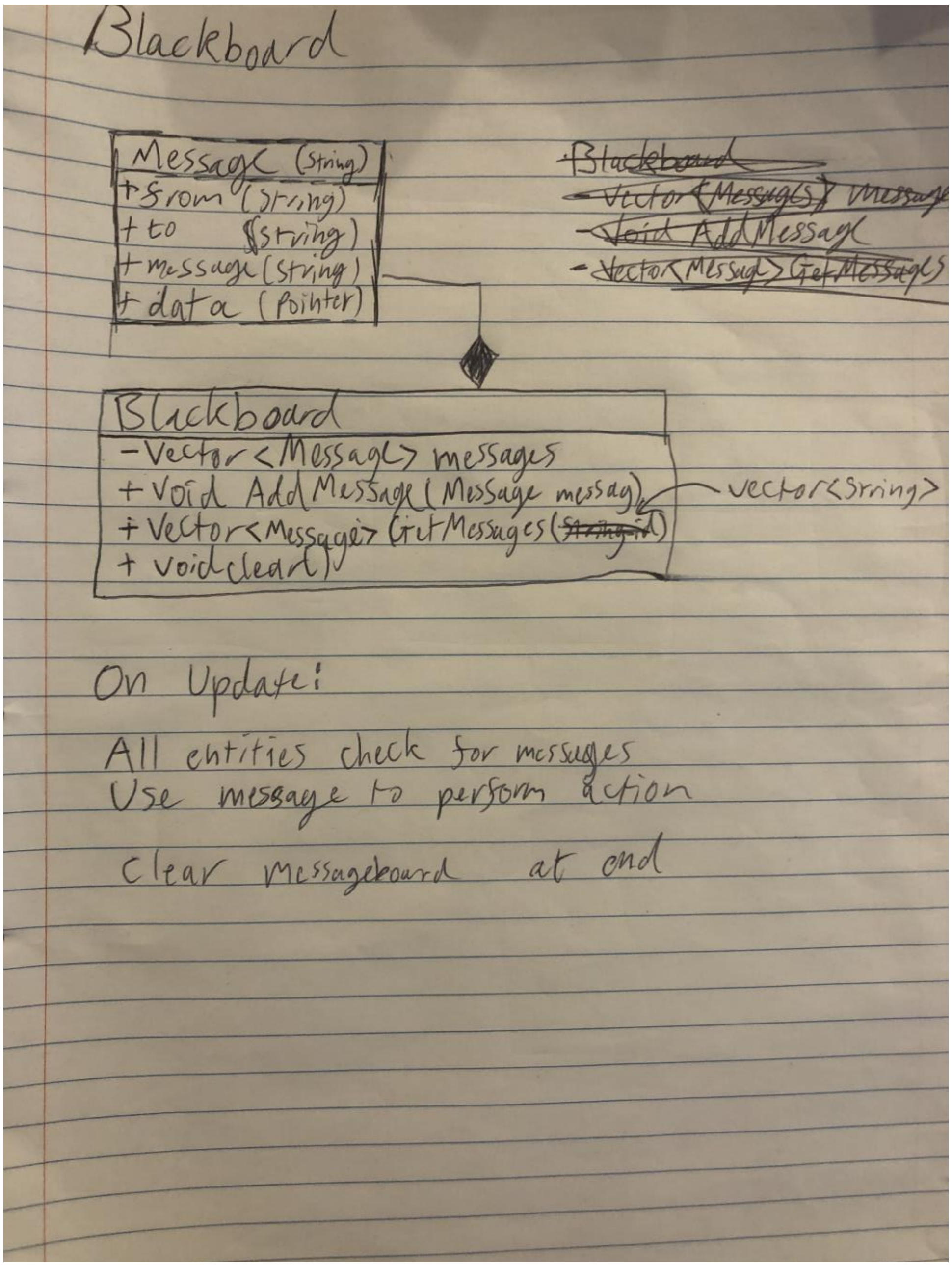
What we found out:

So far in Zorkish related Spikes I have been using a direct approach to adding interactions between entities by calling member functions in commands or in main. In larger games with many more entities a messaging system of some kind is used to avoid having to couple every game object you want to interact.

One approach to a messaging system is a blackboard. Objects in the game can have a reference to an instance of this blackboard in the main game. Each object with a reference to this will be able to send messages to the board that contain 1) who they are from, 2) who they are addressed to, 3) the message or a message id, and optionally 4) any data they want to send/receive.

I made a demo to try and create a simple blackboard that could be used for Zorkish.

First I came up with a desing for how this is going to work.

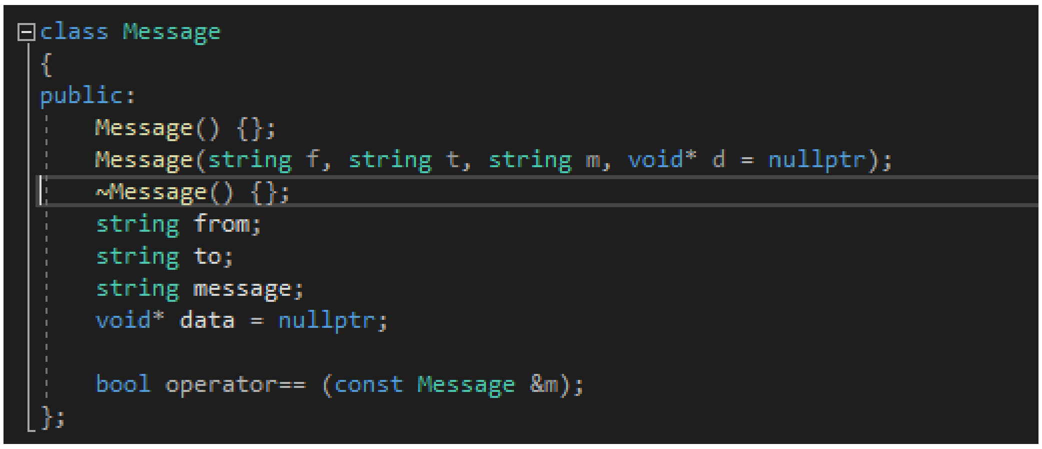

Blackboard
Message (String)
+ from (String)
+ to (String)
+ message (String)
+ data (Pointer)
~~Blackboard~~
~~- Vector<Messages> messages~~
~~- Void AddMessage~~
~~- Vector<Message> GetMessages~~
Blackboard
- Vector<Message> messages
+ Void AddMessage(Message messag)
+ Vector<Message> GetMessages(~~String~~)
+ void clear()
vector<String>

On Update:

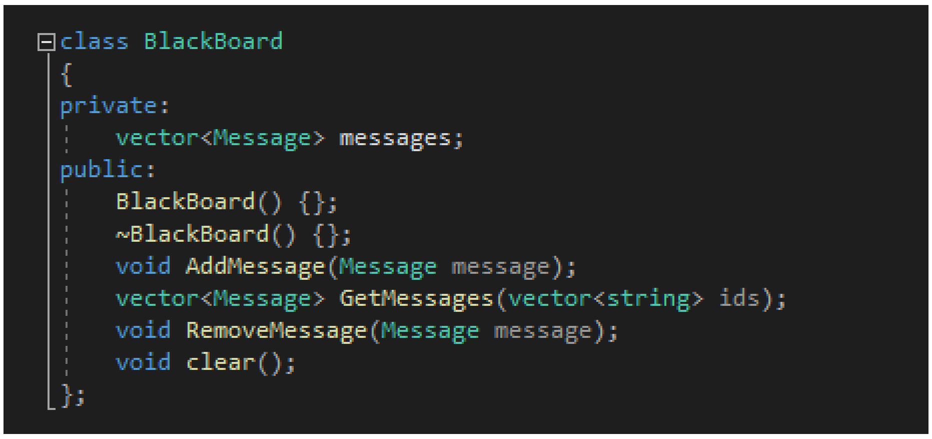
All entities check for messages
Use message to perform action

Clear Messageboard at end

The Blackboard will hold a vector of messages. Objects that have a reference to it can call the AddMessage() function to add a message to the vector. Messages contain 3 strings for the sender, receiver, and message as well as a void pointer to send data. To look for messages, objects with a reference to the blackboard can call GetMessage() and pass in a vector of strings. The function will iterate through the vector to find any messages that contain one of the strings in the vector for the ‘to’ string. Then it will return a vector of messages that match. There is also a function to remove messages which objects can call to remove a message after they are done with it. The clear() function is for use when the program is closed. It frees all of the memory for the pointers in messages just in case the game was closed before the memory could be deallocated.

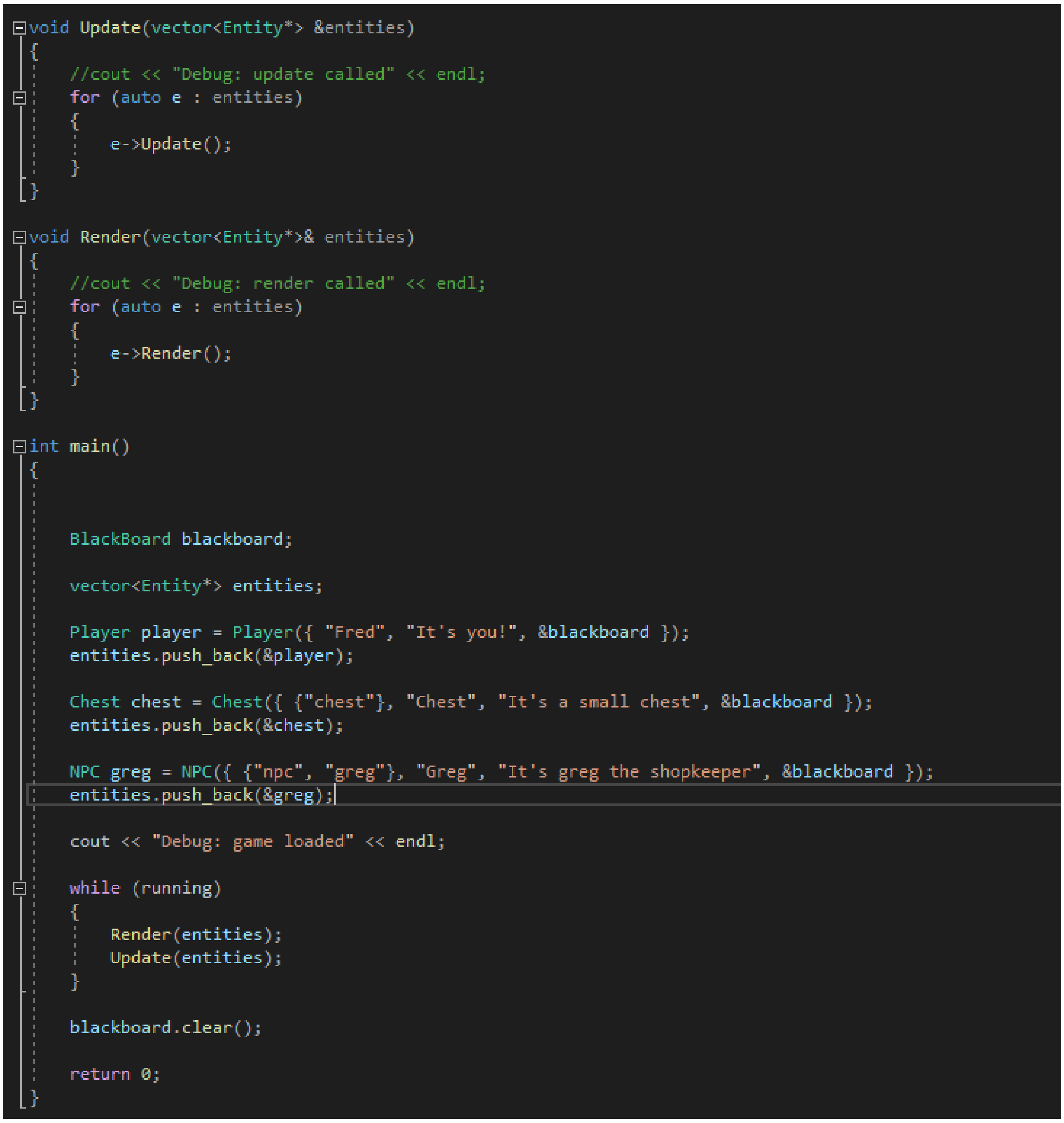


```cpp
class Message
{
public:
    Message() {};
    Message(string f, string t, string m, void* d = nullptr);
    ~Message() {};
    string from;
    string to;
    string message;
    void* data = nullptr;

    bool operator== (const Message &m);
};
```



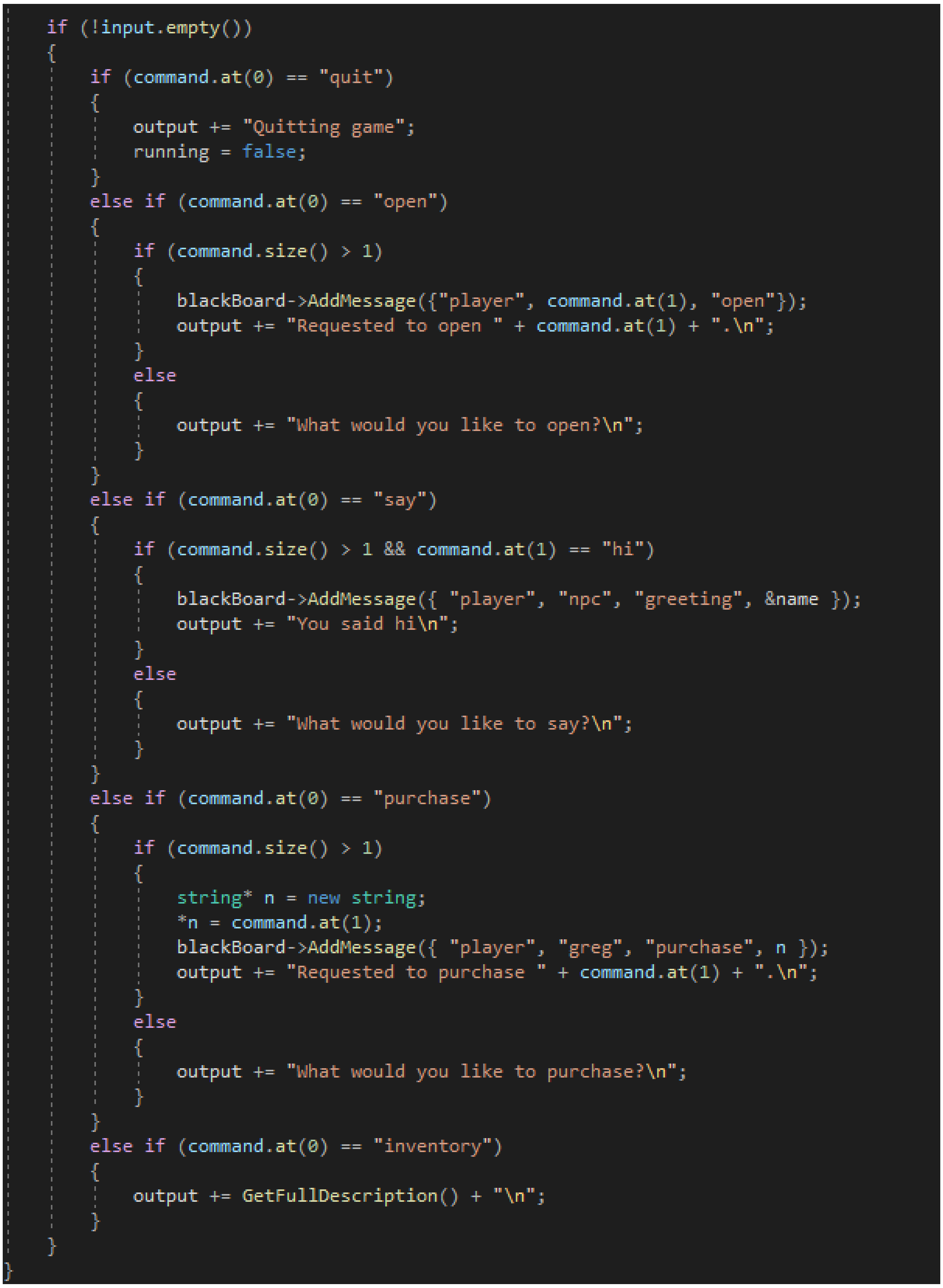
```cpp
class BlackBoard
{
private:
    vector<Message> messages;
public:
    BlackBoard() {};
    ~BlackBoard() {};
    void AddMessage(Message message);
    vector<Message> GetMessages(vector<string> ids);
    void RemoveMessage(Message message);
    void clear();
};
```

I then created three entities to use the blackboard. A player, a chest and an NPC.



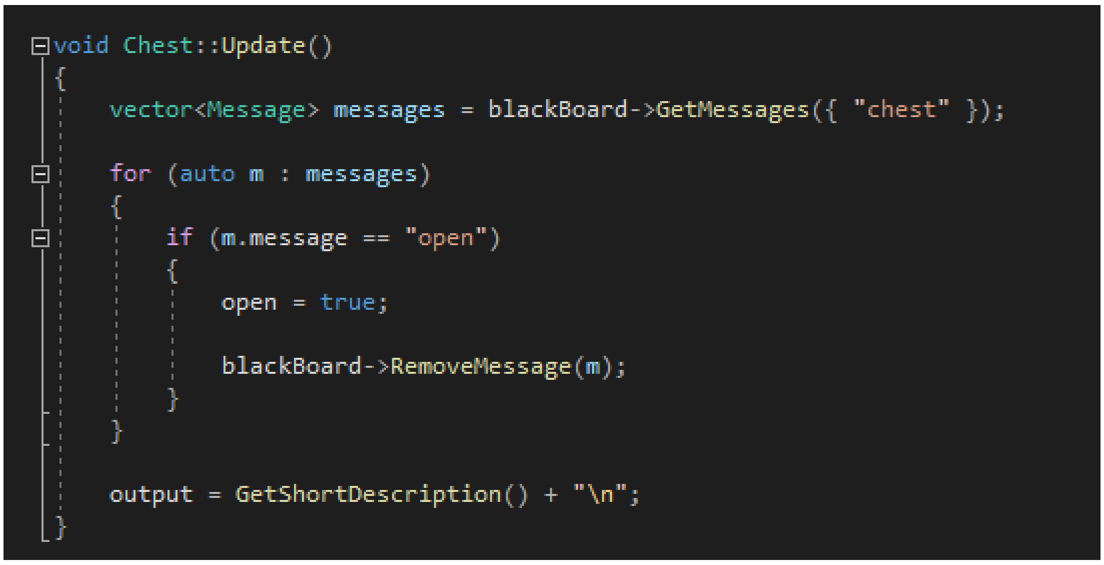
```cpp
void Update(vector<Entity*> &entities)
{
    //cout << "Debug: update called" << endl;
    for (auto e : entities)
    {
        e->Update();
    }
}

void Render(vector<Entity*>& entities)
{
    //cout << "Debug: render called" << endl;
    for (auto e : entities)
    {
        e->Render();
    }
}

int main()
{


    BlackBoard blackboard;

    vector<Entity*> entities;

    Player player = Player({ "Fred", "It's you!", &blackboard });
    entities.push_back(&player);

    Chest chest = Chest({ {"chest"}, "Chest", "It's a small chest", &blackboard });
    entities.push_back(&chest);

    NPC greg = NPC({ {"npc", "greg"}, "Greg", "It's greg the shopkeeper", &blackboard });
    entities.push_back(&greg);

    cout << "Debug: game loaded" << endl;

    while (running)
    {
        Render(entities);
        Update(entities);
    }

    blackboard.clear();

    return 0;
}
```

Every entity has an update and render function which is called in the game loop.



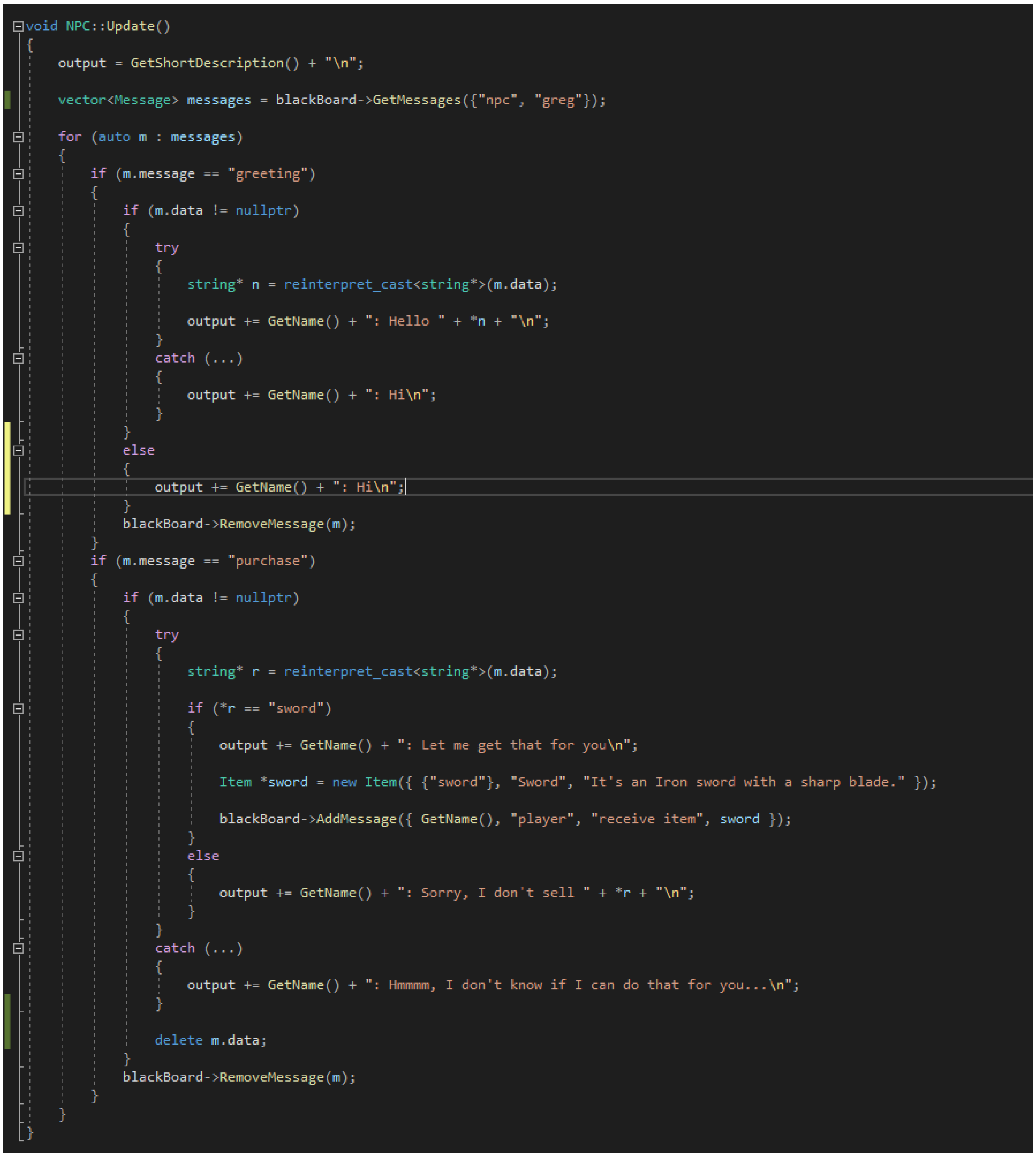
```
    if (!input.empty())
    {
        if (command.at(0) == "quit")
        {
            output += "Quitting game";
            running = false;
        }
        else if (command.at(0) == "open")
        {
            if (command.size() > 1)
            {
                blackBoard->AddMessage({"player", command.at(1), "open"});
                output += "Requested to open " + command.at(1) + ".\n";
            }
            else
            {
                output += "What would you like to open?\n";
            }
        }
        else if (command.at(0) == "say")
        {
            if (command.size() > 1 && command.at(1) == "hi")
            {
                blackBoard->AddMessage({ "player", "npc", "greeting", &name });
                output += "You said hi\n";
            }
            else
            {
                output += "What would you like to say?\n";
            }
        }
        else if (command.at(0) == "purchase")
        {
            if (command.size() > 1)
            {
                string* n = new string;
                *n = command.at(1);
                blackBoard->AddMessage({ "player", "greg", "purchase", n });
                output += "Requested to purchase " + command.at(1) + ".\n";
            }
            else
            {
                output += "What would you like to purchase?\n";
            }
        }
        else if (command.at(0) == "inventory")
        {
            output += GetFullDescription() + "\n";
        }
    }
}
```

The players update function has some basic commands that they can use. Open will send a message to open a chest. If the player uses say and writes hi it will send a message to all npcs that the player said a greeting and pass in the player name as data. The purchase command will request to “purchase” an item from the shopkeeper greg. It will send a string for the item the player wants to buy as data.



```cpp
void Chest::Update()
{
    vector<Message> messages = blackBoard->GetMessages({ "chest" });

    for (auto m : messages)
    {
        if (m.message == "open")
        {
            open = true;

            blackBoard->RemoveMessage(m);
        }
    }

    output = GetShortDescription() + "\n";
}
```

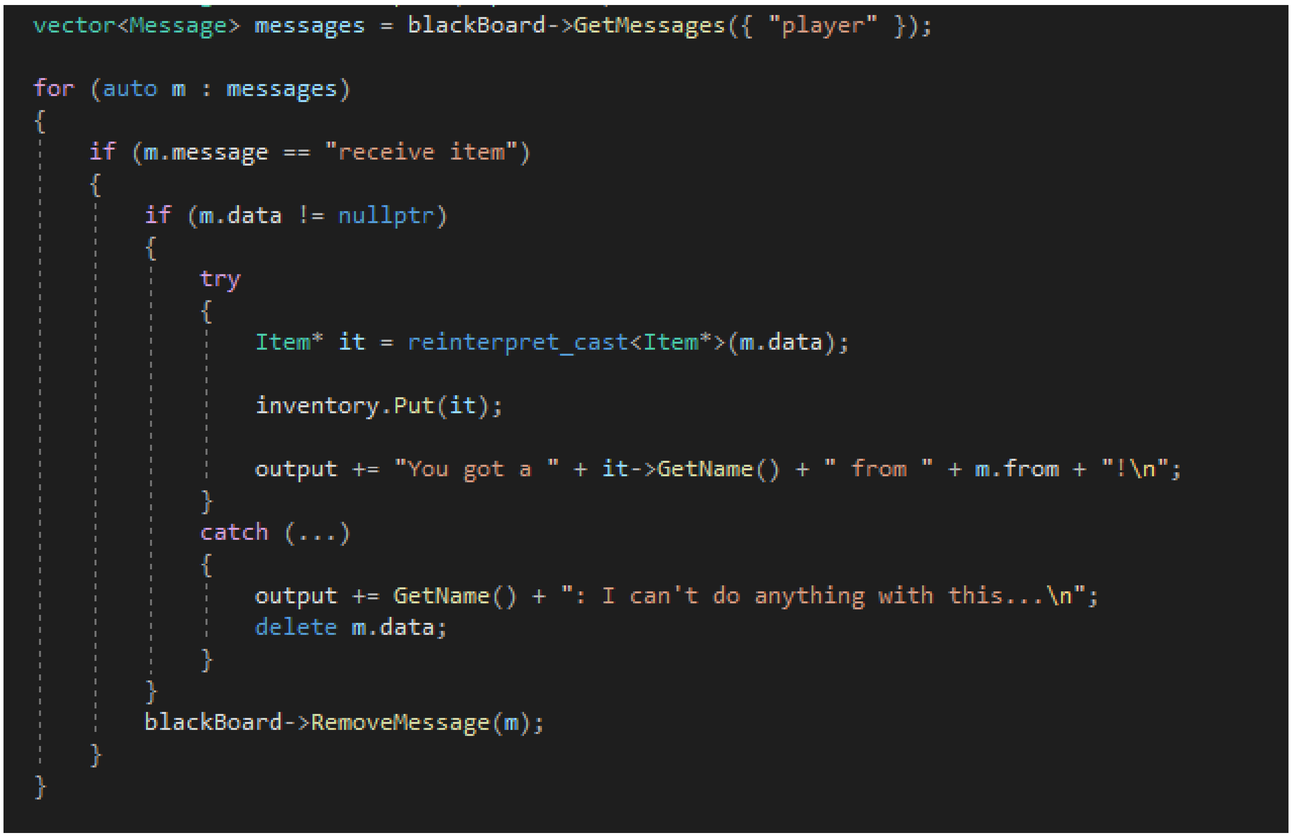
The chest class will search for messages addressed to it. If it finds a message to open it will set its status to open and remove the message.



```cpp
void NPC::Update()
{
    output = GetShortDescription() + "\n";

    vector<Message> messages = blackBoard->GetMessages({"npc", "greg"});

    for (auto m : messages)
    {
        if (m.message == "greeting")
        {
            if (m.data != nullptr)
            {
                try
                {
                    string* n = reinterpret_cast<string*>(m.data);

                    output += GetName() + ": Hello " + *n + "\n";
                }
                catch (...)
                {
                    output += GetName() + ": Hi\n";
                }
            }
            else
            {
                output += GetName() + ": Hi\n";
            }
            blackBoard->RemoveMessage(m);
        }
        if (m.message == "purchase")
        {
            if (m.data != nullptr)
            {
                try
                {
                    string* r = reinterpret_cast<string*>(m.data);

                    if (*r == "sword")
                    {
                        output += GetName() + ": Let me get that for you\n";

                        Item *sword = new Item({ {"sword"}, "Sword", "It's an Iron sword with a sharp blade." });

                        blackBoard->AddMessage({ GetName(), "player", "receive item", sword });
                    }
                    else
                    {
                        output += GetName() + ": Sorry, I don't sell " + *r + "\n";
                    }
                }
                catch (...)
                {
                    output += GetName() + ": Hmmmm, I don't know if I can do that for you...\n";
                }

                delete m.data;
            }
            blackBoard->RemoveMessage(m);
        }
    }
}
```

The npc will also search for messages addressed to it. If it finds a greeting addressed to npcs it will send a reply in the text output using the players name. The void pointer to the data is cast as a string pointer. Just in case we somehow sent the wrong datatype in the message, there is a try and catch statement to handle errors for us. Ideally it should always work as intended but just in case it somehow fails it will catch the error for us and stop the program from crashing.

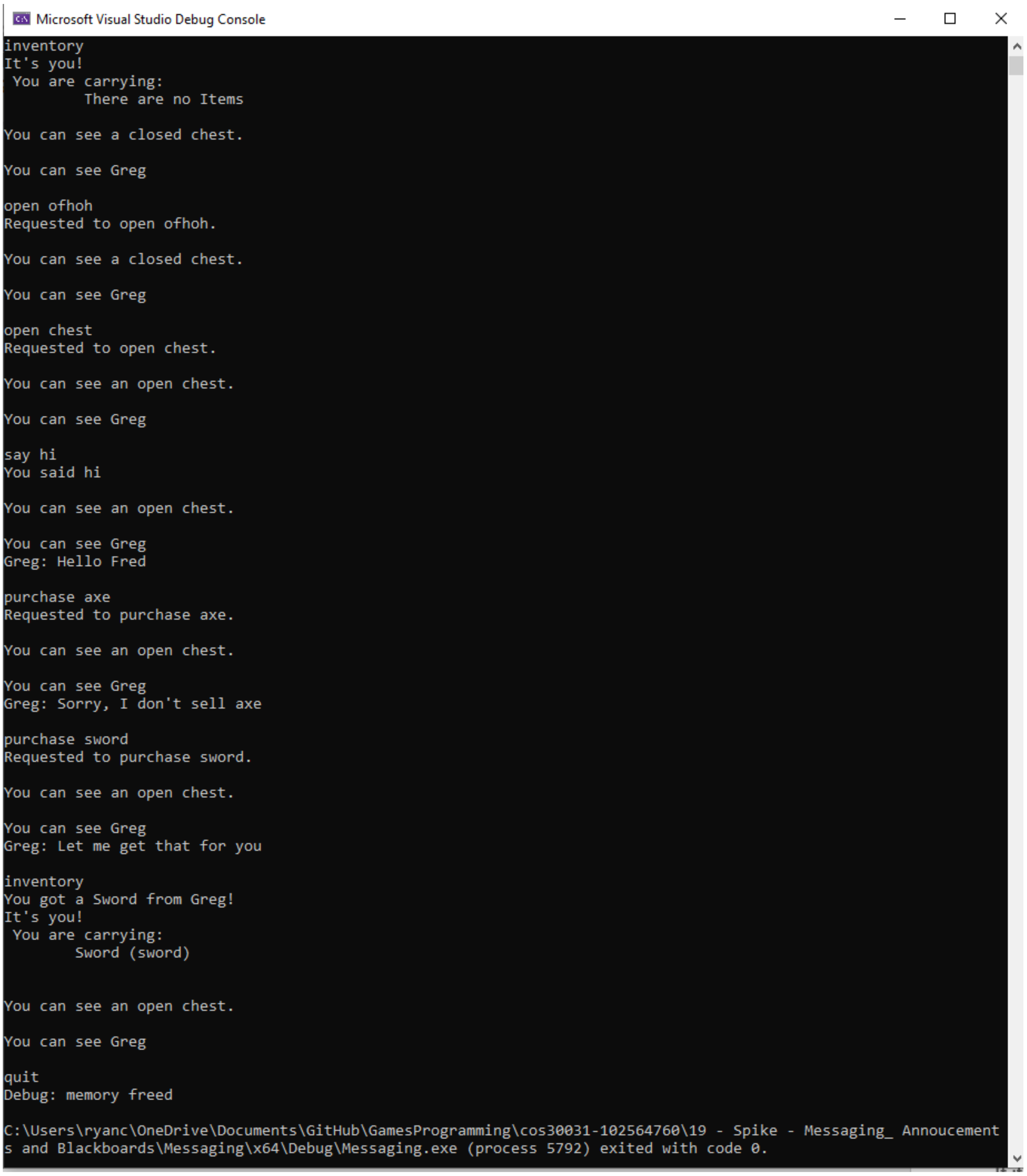
Purchase also works similarly. It casts the void pointer as a string to use. If the player asks for a sword it will create a new Item and send a message to the player to receive the item with the sword passed in as data.



```
vector<Message> messages = blackBoard->GetMessages({ "player" });

for (auto m : messages)
{
    if (m.message == "receive item")
    {
        if (m.data != nullptr)
        {
            try
            {
                Item* it = reinterpret_cast<Item*>(m.data);

                inventory.Put(it);

                output += "You got a " + it->GetName() + " from " + m.from + "!\n";
            }
            catch (...)
            {
                output += GetName() + ": I can't do anything with this...\n";
                delete m.data;
            }
        }
        blackBoard->RemoveMessage(m);
    }
}
```

Back in the player main function if the player gets a message to receive an item it will cast the pointer as an item class then put it in the players inventory. If it fails, the data will be deleted.

Heres what happens when we try and use all the commands.



```
inventory
It's you!
 You are carrying:
         There are no Items

You can see a closed chest.

You can see Greg

open ofhoh
Requested to open ofhoh.

You can see a closed chest.

You can see Greg

open chest
Requested to open chest.

You can see an open chest.

You can see Greg

say hi
You said hi

You can see an open chest.

You can see Greg
Greg: Hello Fred

purchase axe
Requested to purchase axe.

You can see an open chest.

You can see Greg
Greg: Sorry, I don't sell axe

purchase sword
Requested to purchase sword.

You can see an open chest.

You can see Greg
Greg: Let me get that for you

inventory
You got a Sword from Greg!
It's you!
 You are carrying:
        Sword (sword)


You can see an open chest.

You can see Greg

quit
Debug: memory freed

C:\Users\ryanc\OneDrive\Documents\GitHub\GamesProgramming\cos30031-102564760\19 - Spike - Messaging_ Annoucement
s and Blackboards\Messaging\x64\Debug\Messaging.exe (process 5792) exited with code 0.
```