Character AI & Updating Background

Both of the components are integrated into the COMP150 group game. The first is using an API to update the game background based on the location of the ISS. The second is an AI for the character in the game to make it respond to its environment and player input.

Updating Background:

This component uses the Open Notify API to get the location of the International Space Station. Using Open Notify to get a JSON object and the C++ REST SDK, the code extracts the time and the coordinates from the JSON object and moves the background accordingly. The function returns a pplx::task which allows the data to be extracted in the background and not force the program to wait.

```
pplx::task<void> ISSLocation::requestISSLocation()
{ // Sends request to open notify and
    //if successful extracts the JSON data in to a json value
    http_client client(L"http://api.open-notify.org/iss-now.json");
    return client.request(methods::GET).then([](http_response response) -> pplx::task<json::value>
    {
        if (response.status_code() == status_codes::OK)
        { //If connection is successful it extracts JSON
            return response.extract_json();
        }
        return pplx::task_from_result(json::value());
    })
        .then([&](pplx::task<json::value> previousTask)
```

Requests ISS location from Open Notify

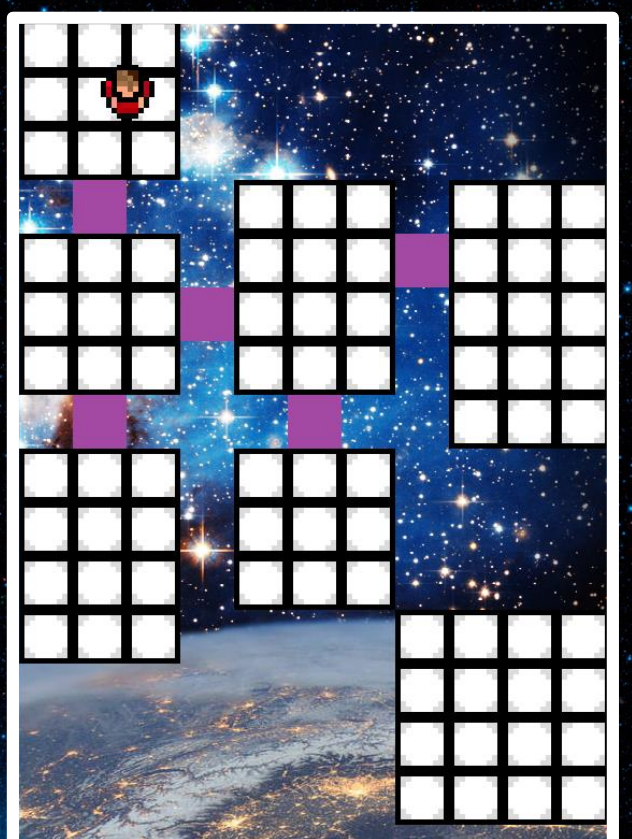
Extracts the JSON data

```
for (auto iter = ISSData.as_object().cbegin(); iter != ISSData.as_object().cend(); ++iter)
{
    const json::value &key = json::value::string(iter->first); //Key is the data name
    const json::value &value = iter->second; //Value is the data itself

    if (value.is_object() || value.is_array())
    {
        if ((!key.is_null()) && (key.is_string()))
        {
            std::wcout << L"Parent: " << key.as_string() << std::endl;
            //Parent is the object in this case iss_position
            //Parent contains the latitude & longitude
        }
        displayJSONValue(value);
        if ((!key.is_null()) && (key.is_string()))
        {
            std::wcout << L"End of Parent: " << key.as_string() << std::endl;
        }
    }
    else
    { //Saves values as variables
        if (key.serialize() == (L"\"timestamp""\""))
        {
            updateTime = value.as_integer();
            // Stores the current time to be used to decide when to update again
        }
        else if (key.serialize() == (L"\"longitude""\""))
        {
            longitude = value.as_double();
            // stores the longitude
```

Saves JSON values as variables

Variables are then used to calculate new X and Y positions for background



Character AI:

The character AI uses the State design pattern to change the character's behaviour depending on the environment.

All the states inherit from CharacterState and using polymorphism they all have an update function. The update function checks the characters health and position to decide which state it should be in. For example if the character's health reaches 0 they enter the DeadState.

```
if (character.health == 0)
{
    character.state = std::make_shared<DeadState>();
    character.isAlive = false;
}
// If the character is in the Idle state
//for too long it changes to the WanderingState
else if (timer > END_IDLE_TIME)
{
    character.state = std::make_shared<WanderingState>();
    character.isWandering = true;
    character.moveCharacter(keyboardState);
}
// If the character has reached the goal the game ends
if (character.reachedGoal(character.getX(), character.getY()))
{
    character.state = std::make_shared<ReachedGoalState>();
}
```

Checks if the character is dead

Checks if the character is ready to start wandering

Checks if the character has reached the end of the level