

Code Analysis Gonk Avoidance

Repo Link: <https://eae-git.eng.utah.edu/u1482733/Gonk-Avoidance>

What Patter(s) did you choose to use?

Why:

I used the command pattern, I used it so that I could easily attach multiple functions to the same command, which allowed me to reuse a key for something entirely different. Implementing this pattern also allowed me to mix and match my key bindings easily.

My Implementation:

```

#include "CoreMinimal.h"
/**
 *
 */
class GONKAVOIDANCE_API Command
{
public:
    Command();
    virtual ~Command();
    virtual void Execute() = 0;
    ATileSelector* tileSelector;
    AGameManager* gameManager;
};

class GONKAVOIDANCE_API MoveUpCommand : public Command
{
public:
    void Execute() override
    {
        if (gameManager->currentMode == Mode::SelectUnit)
        {
        }
        else if (gameManager->currentMode == Mode::Move)
        {
            tileSelector->UnitMovement(Movement::Up);
        }
    }
};

class GONKAVOIDANCE_API MoveDownCommand : public Command
{
public:
    void Execute() override
    {
        if (gameManager->currentMode == Mode::SelectUnit)
        {
            tileSelector->SelectUnit(Movement::Right);
        }
        else if (gameManager->currentMode == Mode::Move)
        {
            tileSelector->UnitMovement(Movement::Right);
        }
    }
};

class GONKAVOIDANCE_API SelectCommand : public Command
{
public:
    void Execute() override
    {
        if (gameManager->currentMode == Mode::SelectUnit)
        {
            tileSelector->SelectionUnit();
        }
        else if (gameManager->currentMode == Mode::Move)
        {
            tileSelector->SelectMovement();
        }
        else if (gameManager->currentMode == Mode::Redo)
        {
            tileSelector->UndoConfirm();
        }
        else if (gameManager->currentMode == Mode::Undo)
        {
            tileSelector->ConfirmMovement();
        }
    }
};

```

This is how I implemented the Command Pattern for my Unreal Project.

What Patter(s), did you deliberately choose not to use?

Why:

I deliberately chose not to completely implement a hierarchy state machine, since the game I was creating didn't really need it. Since the concept for combat is very similar to the game of rock paper scissors and since it is decision based a simple finite state machine is fine, just using Enums for this problem works just fine.

What did you do instead:

I made a simple state machine that uses Enums, in fact there are quite a bit of states that exist in this game apart from just "combat". For example, a list would be combat, movement, turns, color, and more. To be honest I think I over scoped a little bit on this project to the point I had to cut a bit of content for this version at least.

```
10
11 UENUM(BlueprintType)
12 enum class Combat : uint8
13 {
14     None,
15     Attack,
16     Strike,
17     Charge,
18     Defend,
19     Counter,
20     Evade
21 };
22
23 UENUM(BlueprintType)
24 enum class CurrentFileColor : uint8
25 {
26     None,
27     Red,
28     Green,
29     Blue
30 };
31
32 UENUM(BlueprintType)
33 enum class UnitColor : uint8
34 {
35     None,
36     Red,
37     Green,
38     Blue
39 };
40
41 UENUM(BlueprintType)
42 enum class UnitController : uint8
43 {
44     None,
45     AI,
46     Player,
47     PlayerTwo
48 };
49
```

Here are a few of the states that I set up for this project, so that you can get an idea of how I approached this game.

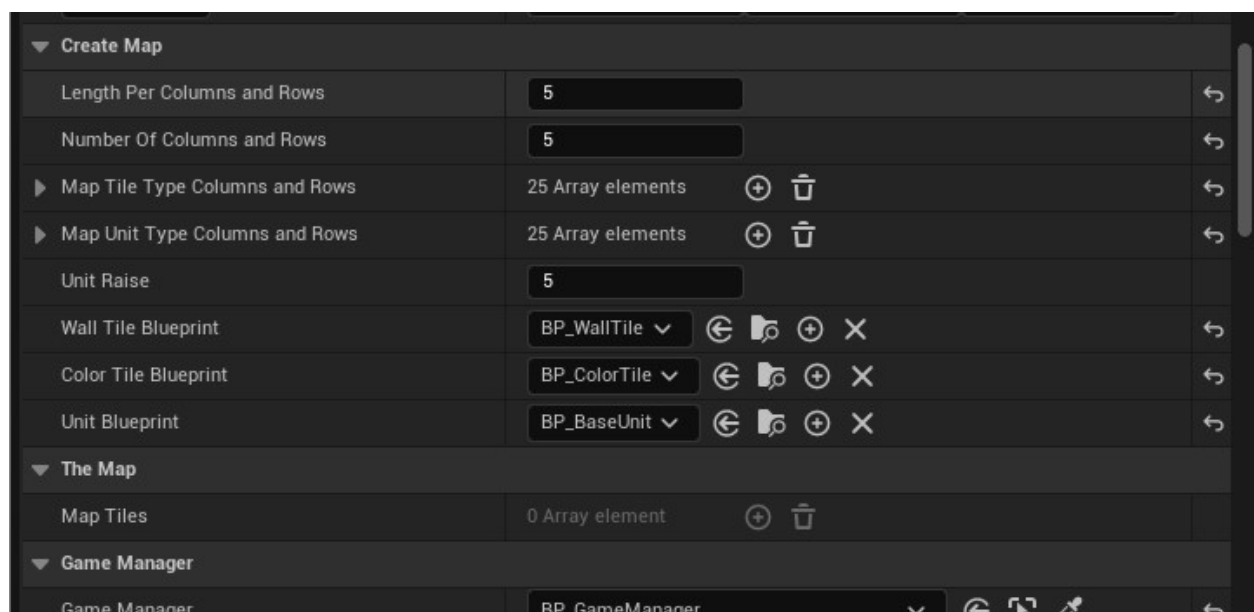
What were the pros/cons of doing it the way you did:

The pros I didn't really have to think about something as complex as a hierarchy state machine, but rather I knew that my states would be a bit more contained to the point I didn't find it necessary to have anything more than just a finite state machine in terms of complexity.

What did you learn from structuring your code the way you did?

Pros: What made it great?

Things that made it great, well to be honest I feel like there is a ton of improvement to be made in this code base alone. I did like how I approached state changes in this project and how I incorporated the command pattern. This was my first time implementing such a pattern and now that I have, I see the potential and all the use cases that may be possible with such a pattern. I did like how I approached my map creator, and I did this so that I could easily generate as many levels as possible. Though I would like to learn more about what types can be visible within the unreal inspector, since I had to use some types that would have benefited more using something else, at least in the way of it being more contained and less complicated.



How could you improve it?

There is something I really want to investigate and that is event calling, I feel like if I just get a better idea of how to do this in code, things will be a lot less complicated to get functioning properly. I had a handful of race conditions that I had to solve, and I feel like if I got more familiar with event-based systems, it would have saved me a lot of trouble.

Cons:

What are the drawbacks?

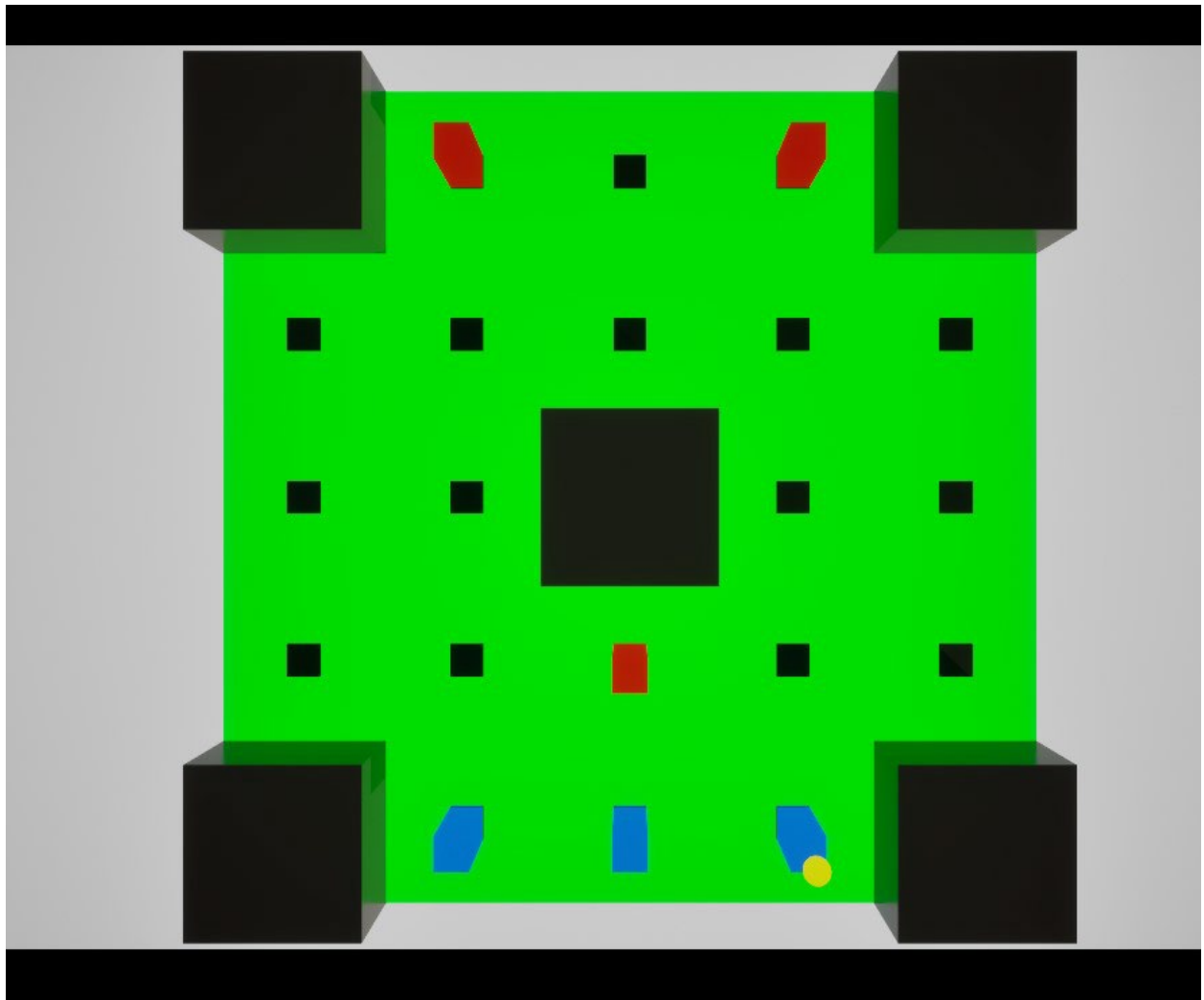
I feel like another drawback of my code is how I incorporate managers, while this works there are more efficient patterns, I am sure of it. Though I ended up doing things the way I did them, due to the time constraints and having to even get a product out.

Did those come from your implementation of the pattern or the pattern itself that didn't really work in this case?

The pattern worked out great! Though it was just my own implementation of some other parts of my code base, though again I would have liked to spend more time making sure that everything is as optimized as possible.

What and how would you do it differently knowing what you do now?

Well, it's more that I would like to research how to build event-based systems, since I feel like that will benefit my own ability in making flexible and usable systems. Though I am happy with what I was able to make and the experience I continue to gather while making games using Unreal Engine and C++.



The Game.