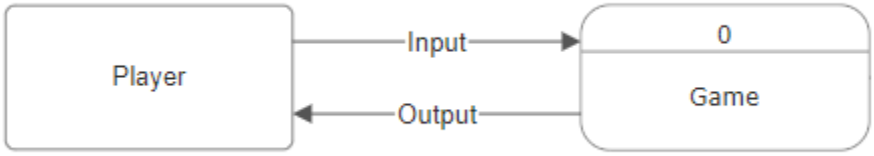
Midnight Slice Madness
Developer Manual
Version 1.0

Environment Setup

1. Ensure your operating system is Windows 11, version 23H2 or higher.
2. Download and install Unity version 2022.3.17f1 from https://unity.com/releases/editor/whats-new/2022.3.17
    a. Further instructions are available at https://unity.com/download
3. Download and install the latest version of git from https://git-scm.com/downloads.
4. Clone the MidnightSliceMadness repository from GitHub using the git command:
   `$ git clone https://github.com/ImNotSebastian/MidnightSliceMadness.git`
5. Launch Unity Hub from the icon on your desktop.
6. Use the “Add” button to navigate to the cloned repository and open the directory at “MidnightSliceMadness/MidSliceMad”
7. Select the now added Unity project to open it.

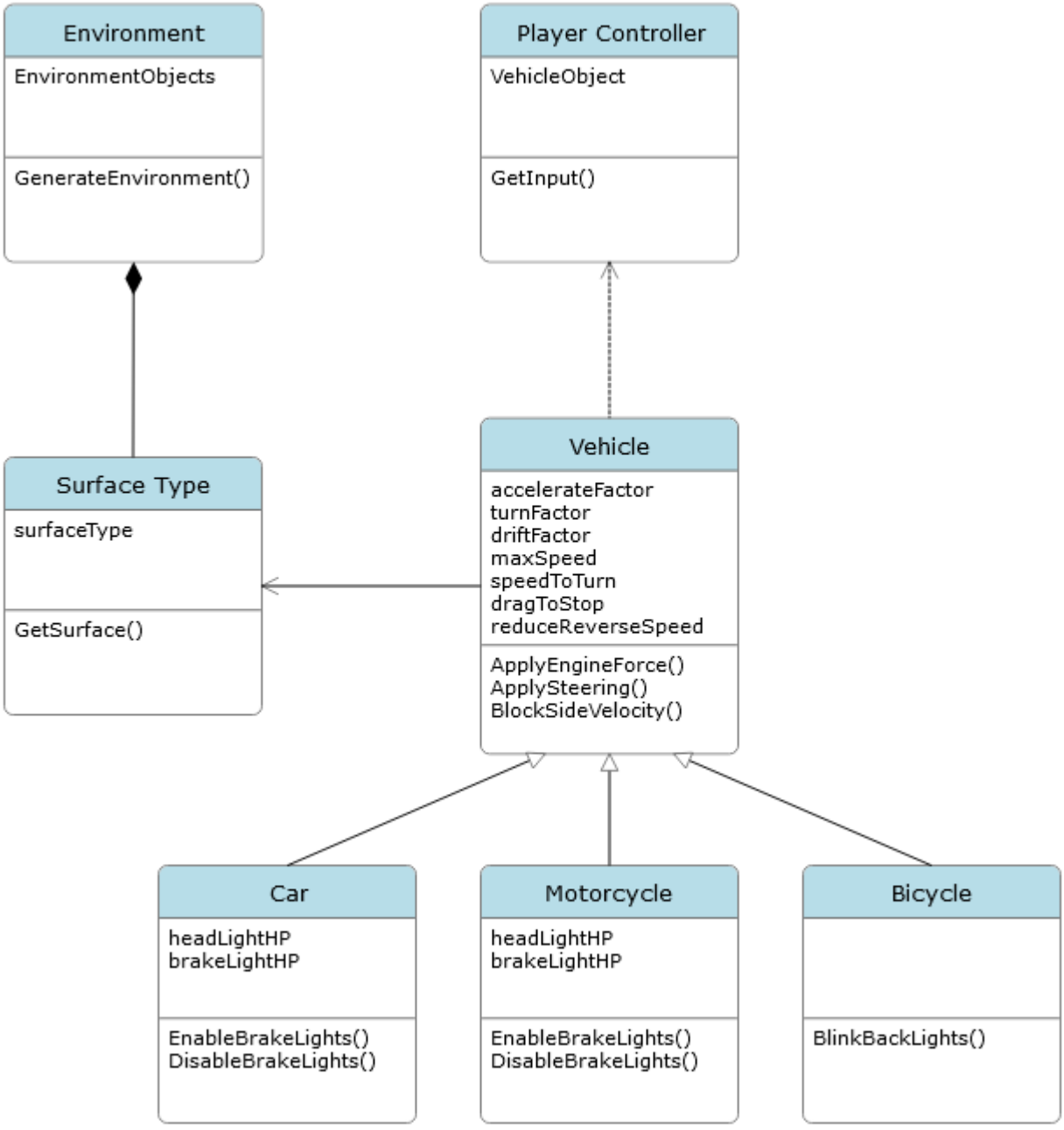
Midnight Slice Madness High-level View

Context Diagram

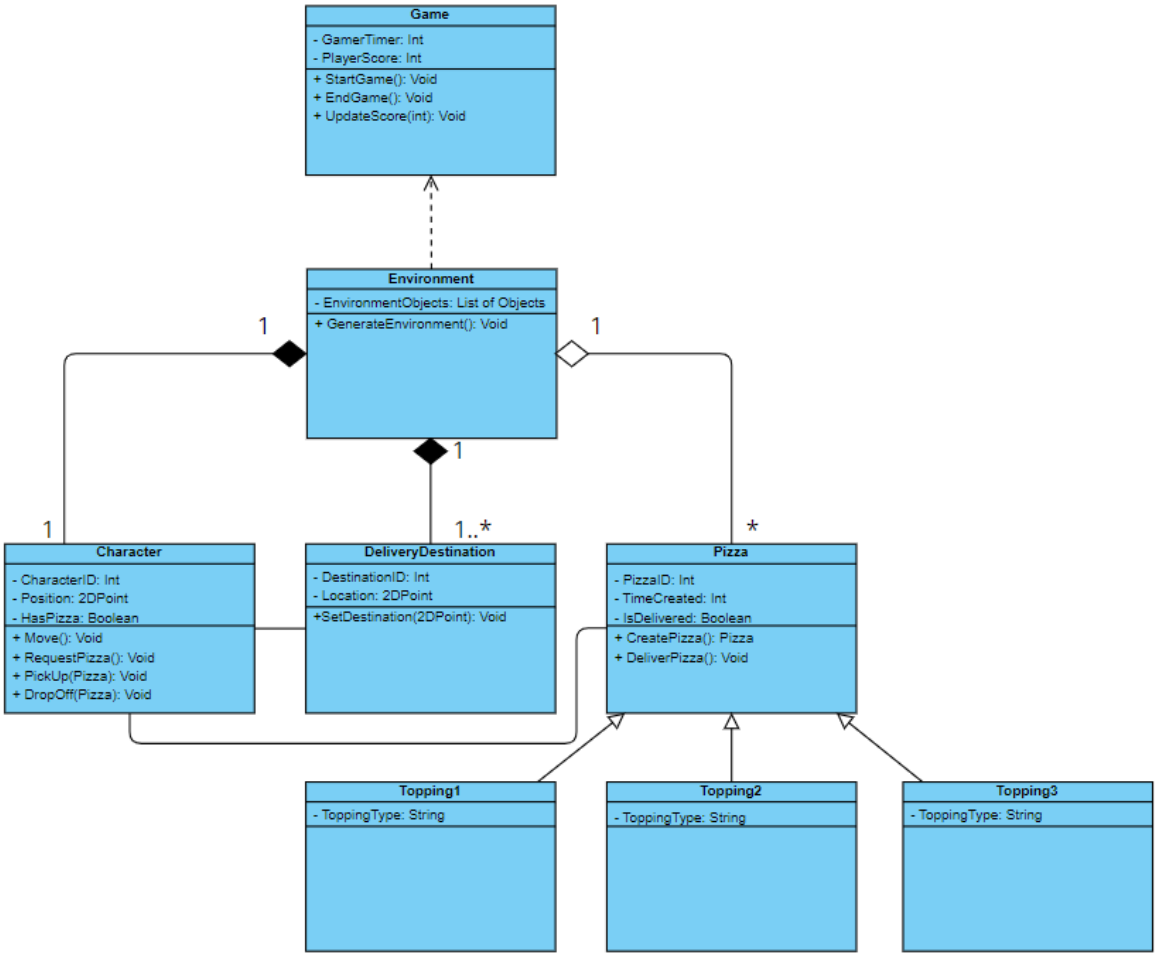


Class Diagrams

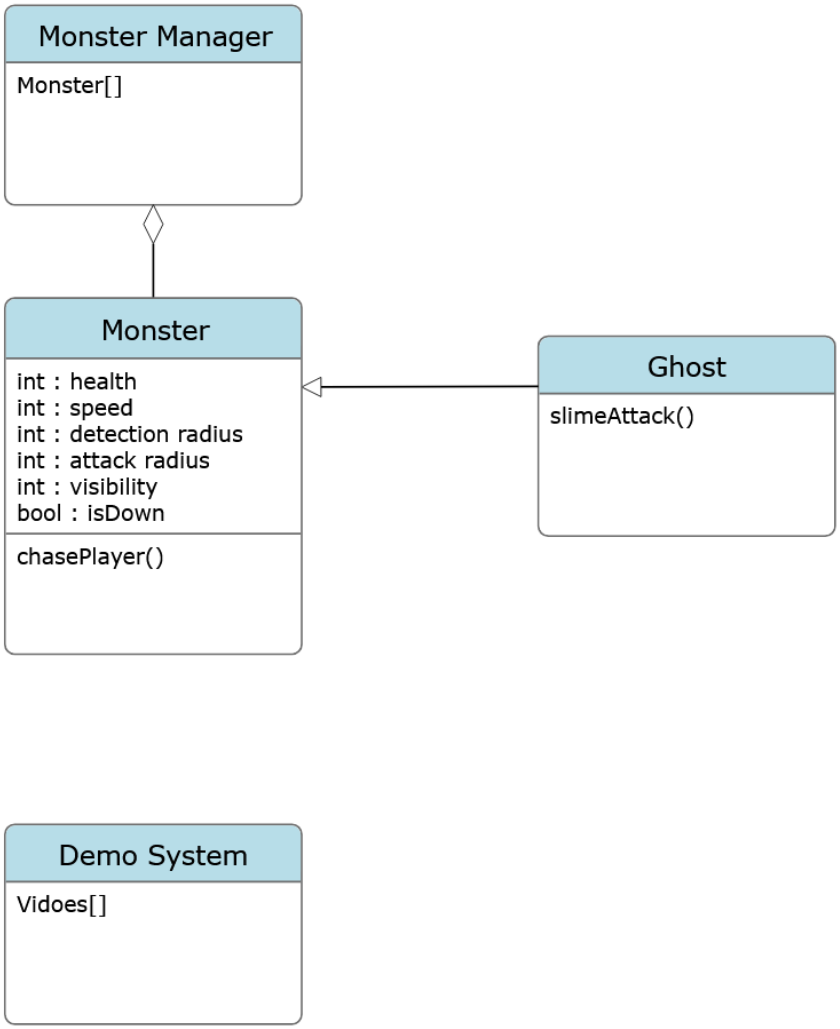
Character and Vehicles



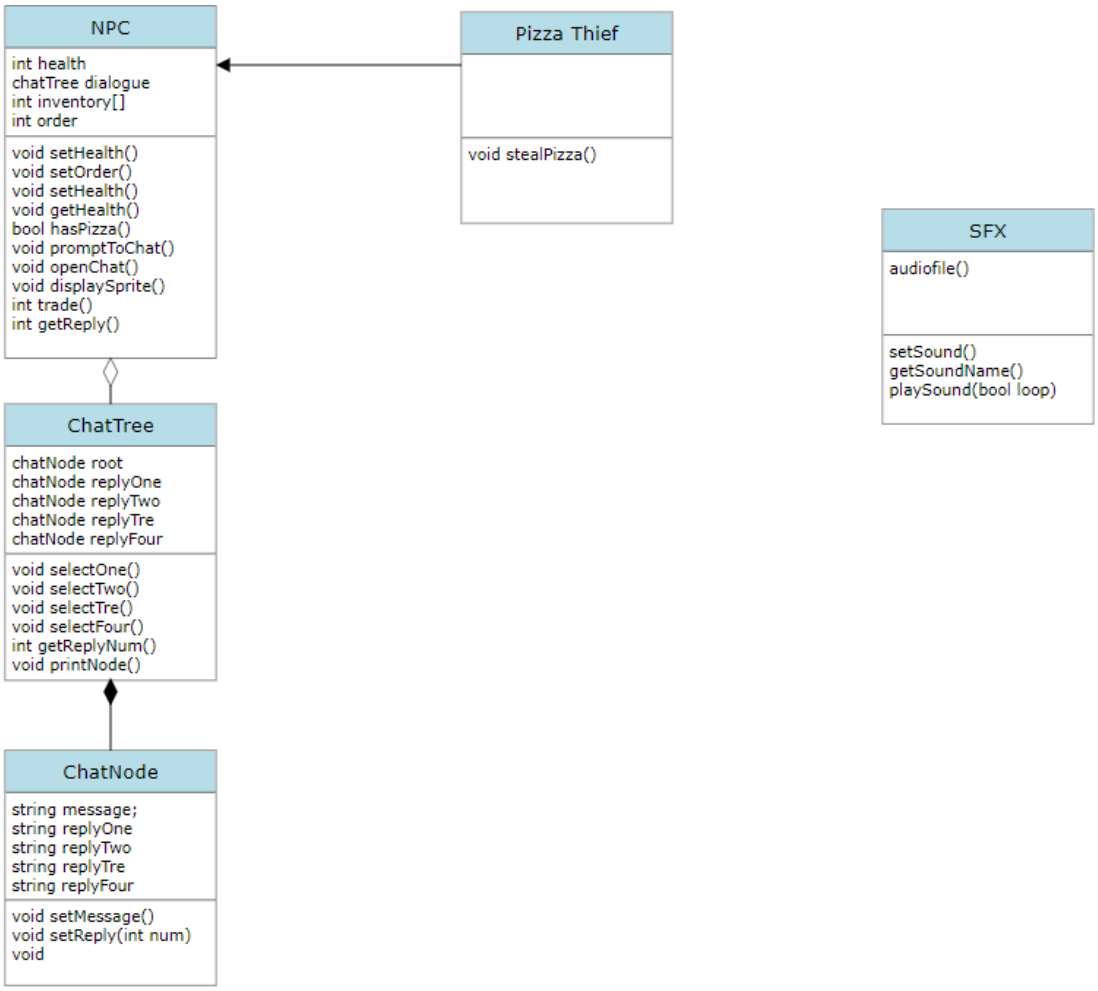
Main Gameplay Loop



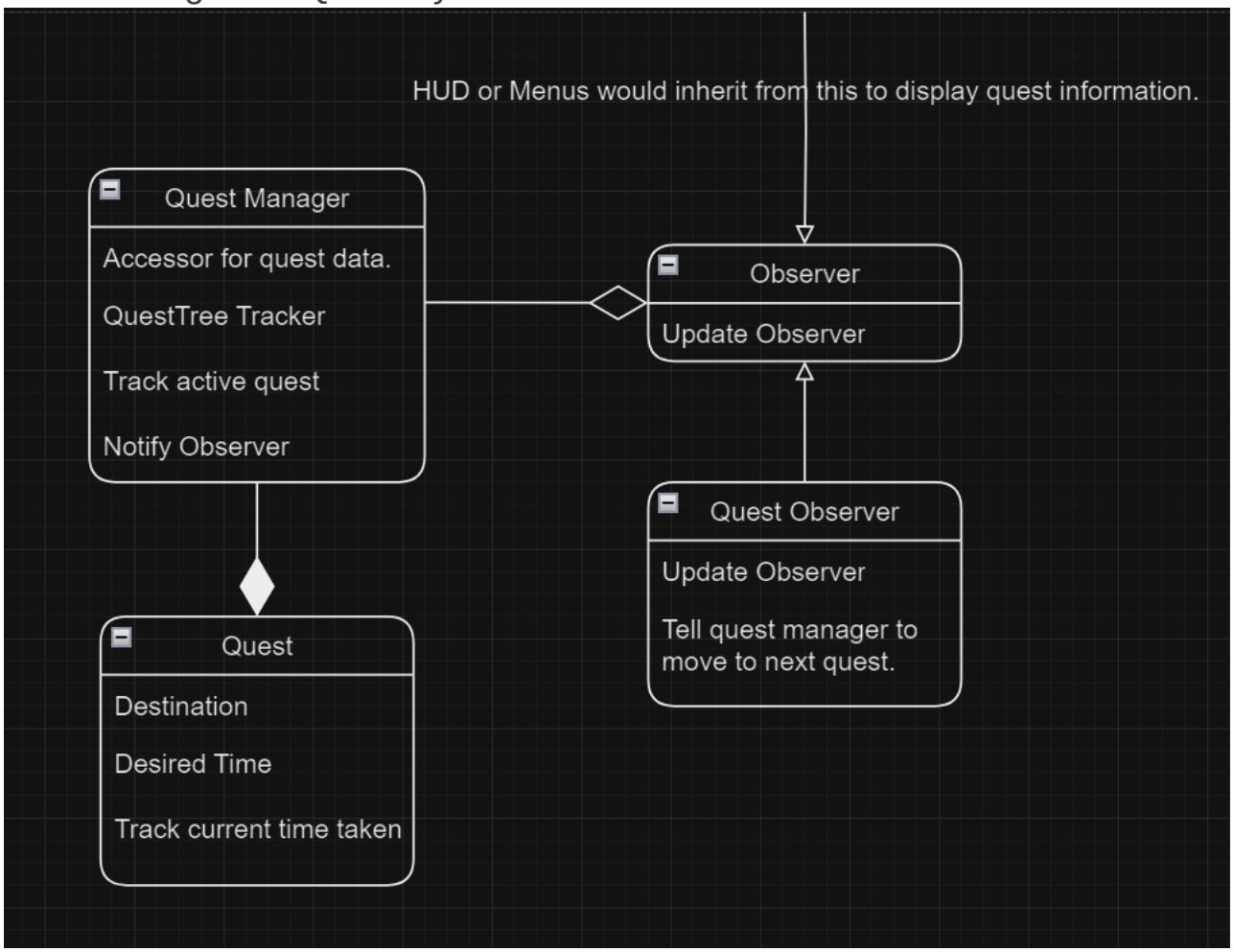
Enemies


Monster Manager
Monster[]
Monster
int : health
int : speed
int : detection radius
int : attack radius
int : visibility
bool : isDown
chasePlayer()
Ghost
slimeAttack()
Demo System
Vidoes[]

NPCs, Chat System, and Sound Effects



Level Design and Quest System



Technical Topics on Oral Exam

Documentation

- Follow the documentation guidelines present in the coding standards document.
- Ensure comments document all prefabs, C# files, classes, and functions.
- Be able to answer the following questions about your documentation:
    - What question are you trying to answer here?
    - Who do you anticipate would be asking that question?

Code Reuse

- Be able to provide an example of reuse in your code.
- Be able to answer the following questions about code reuse:
    - What did you have to do to integrate it with the code you wrote?
    - What are the legal implications if you market your code with the re-used portion?

Test Plan

- Design and implement a comprehensive test plan.
- Provide at least one test case for every non-trivial function.
- Be able to answer the following questions about your test plan:
    - What are you testing?
    - Why did you choose these tests?

Static and Dynamic Binding

- Be able to show a class in your code where there could be either static or dynamic binding.
- Write some mock code showing how you would set the static type and dynamic type of a variable. Choose a dynamically bound method.
    - What method gets called now?
- Change the dynamic type.
    - What method gets called now?
- Pick a statically bound method.
    - Which one would be called in each of the two previous cases?

```
#include <iostream>
using namespace std;

class Grandpa
{
   public:
     virtual void DoDynamic() {cout << "      Dynamic Grandpa" << endl;}
     void DoStatic(){cout << "      Static Grandpa" << endl;}
     void DoPartial() {cout << "      Partial Grandpa" << endl;}

};

class Father: public Grandpa
{
   public:
     void DoDynamic(){ cout << "      Dynamic Father" << endl; }
     void DoStatic(){cout << "      Static Father" << endl;}
     virtual void DoPartial() {cout << "      Partial Father" << endl;}
};

class Son: public Father
{
   public:
     void DoDynamic(){ cout << "      Dynamic Son" << endl; }
     void DoStatic(){cout << "      Static Son" << endl;}
     void DoPartial() {cout << "      Partial Son" << endl;}
};
```

Software Patterns

- Review the team lead 3 presentation on software patterns if needed (located in the “doc” folder of your GitHub repository.

- You can find a list of software patterns with class diagrams here: https://sourcemaking.com/design_patterns
- Choose and implement at least one big pattern or two small patterns (singleton and private class).
    1. Read the problem section of each pattern.
    2. Determine if the problem is present in your code.
    3. If so, read the example section and confirm it is like your situation.
    4. Choose the pattern.
- Ensure they are relevant and make sense to use for your specific problem.
- Be able to answer the following questions about patterns:
    - Which patterns did you choose?
    - Why did you choose each pattern?
    - Is the use of each pattern justified?
- Choose the pattern you know best. Be able to answer and do the following:
    - Would something else have worked as well or better than this pattern?
    - When would be a bad time to use this pattern?
    - Draw the class diagram for it.

Creating A Prefab

- Review the team lead #2 presentation document in the section on prefabs if needed (located in the “doc” folder of your GitHub repository).
- Prefabs allow you to store a GameObject object with its components and properties as a reusable asset.
- New prefab instances can be created from a prefab asset.
- Useful for frequently occurring elements like characters or scenery.
- Creating a prefab is straightforward:
    1. Select **Asset > Create Prefab** in the Unity Editor.
    2. Drag an object from the scene onto the “empty” prefab asset that appears.
    3. Drag the prefab asset from the project view to scene view to create instances of the prefab.
    4. There are now instances of a prefab in your game.
- Helpful resources for more information:
    - https://docs.unity3d.com/2023.2/Documentation/Manual/Prefabs.html
    - https://docs.unity3d.com/2023.2/Documentation/Manual/CreatingPrefabs.html
    - https://ouzaniabdraouf.medium.com/how-to-create-prefabs-in-unity-8d2ff87bdad6