Report

Changes made to the project and Design analysis

(P1) Design of editor:

Game Launch

- The original project PacMan in the TorusVerse consists of two parts, one is 2D Map Editor master, and the other is the game main program called pacman. We combined the two parts together to create a complete game with editor functionality.
  In detailed, We have modified the Driver class. Depending on the running program arguments, we can activate edit mode and test mode separately by calling controller() with different parameters.

Map Reading and Saving(Adapter Pattern)

- We abandoned the original function of using the PacManGameGrid class to obtain grid maps, and instead used the MapCodeImp call decorator to read map information from an XML file.
- In the future, we may need to handle more types of map files. If we write the reading methods of map files in different formats in the same method, it will make lengthy code difficult to maintain and call. Therefore, using adapter allows us to write different adapters to handle different formats of map files. Convert them into a format that can be uniformly processed by MapCodeImp
- The original editor saves map information in an XML file format, while the tester reads the background and actor settings of the "PacManGameGrid" and configuration file ". properties". The data formats of the two programs are different, and the adapter design mode is used. When reading XML, the "MapCodeAdapter" is used to convert the data format, allowing Test to load the XML file without changing the original program.

```
└── xmlToTest
    ├── MapCode.java
    ├── MapCodeAdapter.java
    └── MapCodeImp.java
```

- In MapCode, we use the decorator mode in design mode to dynamically adjust the way map information is read based on the type of map format being read. By implementing the MapCode interface in the MapCodeAdapter, we can achieve map reading functionality in a more independent manner. If future development requires us to be able to read maps in other formats, we only need to create a new adapter to dynamically change the way we read maps, rather than directly adding a new method of reading maps in MapCodeImp. This can add or remove read types in a more flexible way. It is conducive to reducing the coupling of code. When other classes want to use these decorators to read map information, they can also be called directly without calling methods in MapCodeImp. Improved code reusability
  In the future, we may need to handle more types of map files. If we write the reading methods of map files in different formats in the same method, it will make lengthy code difficult to maintain and call. Therefore, using adapter allows us to write different adapters to handle different formats of map files. Convert them into a format that can be uniformly processed by MapCodeImp

Level Check and Game Check(Factory Pattern)

- Regarding map checking and game checking, as our game contains many game rules and map restrictions, simply writing all the check rules into one file may make the code confusing and difficult to maintain, especially when the number of rules increases or the rules become complex. In contrast, using the factory mode has the following advantages:
- Decoupling: The factory pattern separates the creation and use of checking rules. This makes the code more flexible, as you can change the creation logic of rules without affecting the code that calls game checks.
- Extensibility and code reuse: When you need to add new rules, simply get the already written rules in GameCheckFactory and LevelCheckFactory and add them one by one to our checklist. This way, your code can easily extend the new rules and freely call the required check rules
- Encapsulation: The factory mode encapsulates the creation process of checking. If the process of creating rules requires some complex steps or dependencies on other objects, these details can be encapsulated in the factory, making the process of creating rules transparent to external code.

LevelCheck checks when saving and editing maps in the editor. Each check condition and corresponding Log print message exist in a class, implementing the interface "LevelCheck"

```
├── check
│   ├── gameCheck
│   │   ├── GameCheck.java
│   │   ├── GameCheckFactory.java
│   │   ├── GameChecking1.java
│   │   └── GameChecking2.java
│   └── levelCheck
│       ├── LevelCheck.java
│       ├── LevelCheckFactory.java
│       ├── LevelChecking1.java
│       ├── LevelChecking2.java
│       ├── LevelChecking3.java
│       └── LevelChecking4.java
```

- And it will be produced by the factory class "LevelCheckFactory". If you need to add inspection conditions and Log information in the future, create a new class, and implement the LevelCheck interface.
- The implementation method of GameCheck is the same as LevelCheck, but GameCheck checks before loading the map after pressing the Test button.

```
public class LevelCheckFactory {
    public static LevelCheck getLevelChecking(int check, GameCallback gcb) {
        LevelCheck lc = null;
        switch(check) {
            case 1:
                lc = new LevelChecking1(gcb,"");
                break;
            case 2:
                lc = new LevelChecking2(gcb,"");
                break;
            case 3:
                lc = new LevelChecking3(gcb,"");
                break;
            case 4:
                lc = new LevelChecking4(gcb,"");
                break;
        }
        return lc;
    }
}
```

(P2) Design of auto-player:

Auto Play(Strategy Pattern)

- We have refactored the original autoplay method by using policy mode to flexibly change the functionality of autoplay and rewritten an autoplay method, allowing the new autoplay to use transport gates to reach enclosed spaces and eat the balls

Considering the future of the game, we may introduce new features and mechanisms, and AI's strategy should also be adjusted at that time. To test new features, we may need to use different search algorithms to test autoplay. The logic of these methods is similar in structure, but different in implementation. If we change the content of autoplay directly or add and change the content of autoplay in the original PacActor, the operation will become very cumbersome, and the coupling of the code will be increased. Putting all autoplay in PacActor is also not conducive to our switching and comparison between different autoplay mechanisms. Therefore, we adopted a Strategy Pattern to encapsulate different types of autoplays. By using policy patterns, we can avoid using a large number of conditional judgment statements to call different autoplays in PacActor, making the code easier to understand and maintain. Allocating the code that implements the autoplay function to the autoplay strategy to deal with it separately also reduces the coupling of the code and greatly facilitates the switching of different autoplay methods.

-Use strategy mode to switch algorithms

-To add a new AutoPlay algorithm, create a class that implements "AutoPlay" in the "AutoPlay" folder and override the "moveInAutoMode()" method. When creating AutoPlay in Game, make the newly created AutoPlay point to the newly written class.

```
public class AutoPlay2 implements AutoPlay{
    private Game game=null;
    private PacActor pacActor=null;
    private Random randomiser=null;

    public AutoPlay2(Game game, PacActor pacActor, Random randomiser) {
        this.game = game;
        this.pacActor = pacActor;
        this.randomiser = randomiser;
    }
    @Override
    public void moveInAutoMode() {

    }
}
```

How to adjust assuming that eating an ice cube freezes the monsters for a defined duration:

- If eating a piece of ice will freeze the monster within the specified time, we can implement AutoPlay by creating a new call: autoplay3. We can add the operation that the player makes when touching the ice cube in autoplay 3 to synchronize the monster's movement information on the map, and call this new autoplay 3 instance in PacMan class to enable the new autoplay method to adapt to the new ice cube function.

(P3) Additional design:

1. Jump to the next level

- After finishing all the balls and gold in the current level, if it is not the last digit at the beginning of the file name, it will skip to the next level to continue running.
- How to achieve this: When reading a file, save the abstract paths of all XML files that meet the naming requirements in the file directory, and then execute them in sequence. If one level is not completed, loop to the next level

```
private boolean LevelChekMap() {
int checkNum=0;
for(int i=0;;i++) {
LevelCheck levelCheck = LevelCheckFactory.getLevelChecking(i+1,gameCallback);
if(levelCheck==null) break;
if(levelCheck.CheckCondition(model.getMap())==0) checkNum++;
}
if(checkNum!=0) return false;
return true;
}

private boolean gameCheckMap(File parentFile) {
int checkNum=0;
for(int i=0;;i++) {
GameCheck gameCheck = GameCheckFactory.getGameCheck(i+1,gameCallback);
if(gameCheck==null) break;
if(gameCheck.CheckCondition(parentFile)==false) checkNum++;
}
if(checkNum!=0) return false;
return true;
}
```

2. Portal

- The class used in manual or automatic mode: ' PortalFunction '
- before refreshing the next position, We firstly determine whether the next position is the position where the conveyor door is located. If so, assign the next position' Location 'to the corresponding position of the conveyor door in another position to achieve the transmission effect.

```
public Boolean isPortal(int x,int y) {
    for(int i=0;i<4;i++)
        for(int j=0;j<2;j++)
                if(Portal[i][j][0] == x && Portal[i][j][1]==y) return true;
    return false;
}

public Location getNetLocation(Location now) {
    Location next = now;
    int x = now.getX();
    int y = now.getY();
    for(int i=0;i<4;i++)
        for(int j=0;j<2;j++)
            if(Portal[i][j][0] == x && Portal[i][j][1]==y) {
                next = new Location(Portal[i][1-j][0],Portal[i][1-j][1]);
                return next;
            }
    return next;
}
```

3. Autoplay mode switch

- In addition, we have added a function to manually switch between automatic programs in the test, allowing players to switch to autoplay when they press the "enter" button and exit autoplay when they press the "space" button.