Pandemic

CSSE 375 - Software Constraints and Evolution

1/20/2016

Chris Budo, Andrew Ma, Austin Fahsl, Gene Logan
Milestone 4

Table of Contents

*Pandemic official rulebook: https://goo.gl/CcLs10

Chris Budo, Andrew Ma, Austin Fahsl, Gene Logan
Milestone 4

1 Description

Pandemic is a cooperative game played by 2-4 people. The goal of the game is to fight the spread of a disease as it spreads across the globe while simultaneously creating a cure for the disease. The project currently exists as a windows forms application, throughout the course of CSSE 375, we will be going about updating and refactoring the code to allow for more easy extension, simplify unnecessarily complicated logic, and elimination of 'bad smells'.

2 Communication

We will be meeting weekly with the instructor where we will give him updates about the progress on the project, progress will also be tracked through commits and issue tracking on github and Waffle.io the intention is that members of the project will make commits reguarly as they finish tasks. The team will be communicating through a google hangout and the school provided email. For tracking the work that is being done and has been done, we will be using a service called Waffle.io which will allow us to create a kanban board consisting of issues that exist in our github project, we are using this over options such as Trello or Taiga because we feel that this will be more easily utilized as we progress through our project.

3 Project Access

Github: https://github.com/cbudo/Pandemic-Windows
Task-board: https://waffle.io/cbudo/Pandemic-Windows
(to join: https://waffle.io/cbudo/Pandemic-Windows/join)

4 Project Deliverables

By the end of the term, we will deliver the project in the form defined by the following requirements:

1. Basic Pandemic game complete with all rules and features described in the rulebook* for the original set.

2. Addition of difficulty settings as per the difficulty descriptions in the original rulebook*.

3. Various quality of life improvements to gameplay, including but not limited to highlighting a player's possible moves and the inclusion of help panels.

*Pandemic official rulebook: https://goo.gl/CcLs10

4. Integration of features from the State of Emergency expansion pack.

5 Bad Code Smells

5.1 Milestone 2

1. **Long Class/Divergent Change** - GameBoardModels.cs - Chris
   There are multiple classes inside of the GameBoardModels class, which do not seem essential to the purposes of the class. In order to fix this, we plan on extracting these classes from GameBoardModels and refactoring anywhere the values are used.

2. **Data Class** - GameBoardModels.cs - Chris
   GameBoardModels is the class in the project where all of the data is stored. We plan to change this by moving the variables into the classes that they match better and creating better ways of accessing them from other classes they are necessary for.

3. **Duplicate Code** - PlayerActionsBL.cs - Austin
   The PlayerActionsBL class has several functions that are extremely long due to repeated chunks of code. We plan on extracting these chunks of code to private methods to reduce the length of the class and make each function easier to read.

4. **Comments** - Throughout entire project - Gene
   Excessive comments are present in most of the project files. In many instances, they have no purpose because the simplicity of the code is already sufficient for understanding how it works. In other cases, comments serve to explain a method or class with a name that poorly conveys its purpose.

5. **Duplicate Code** - Create.cs - Austin
   The Create class manages the creation of cities, cards, and other resources. Currently, this creation is done in a very sloppy way, with each line hard-coded with set strings. This should be changed to pull from a resources file and use a loop to generate the cities, cutting down up to a hundred lines of code.

6. **Empty Classes/Methods** - Extraneous locations, including InfectorWS.cs - Andrew
   In each class, there are areas where there are dead code. There are if/switch statements with no logic in them. For the if/switch statements, the goal is to reorganize the logic in order to have each case have something. For empty classes/methods, the goal is just to remove them.
7. **Feature Envy** - HelperBL.cs - Gene
   HelperBL is full of utility methods that deal with shuffling card data, but are declared as static because the class doesn't actually encapsulate any data. To solve this, a Deck class will be created to encapsulate multiple card objects and perform

various functions on them such as the shuffling that is included in this class.

8. **Divergent Change** - Card.cs - Andrew
   We will fix the divergent change bad smell with the Strategy Pattern. We will create a CardBehavior Interface which will be implemented by specific card types.

5.2 Milestone 3

1. Change the win/loss InvalidOperationException structure to use custom exception types and change the win logic to the cure class, since the only way to win involves changing the state of the cures. - Austin

2. **Long Method/Switch Case** - InfectorBL.cs InfectCity method is huge. This can be fixed by creating private helper functions and reducing duplicate sections of the method. - Gene

3. Reorganize DispatcherMove.cs code into proper class. - Chris

4. **Duplicate Code** - City.cs SetDiseaseCubes() has duplicate code that needs to be extracted into private functions.
   Andrew

5. Organize the project files into appropriate packages. Right now, most of them are just in the root directory and navigating the project is clunky. - Austin

6. **Primitive Obsession** - Create.cs MakeInfectionDeck() - This deck is built out of a string array instead of an encapsulated structure that has important functions like "shuffle."

6 Refactor Analysis

6.1 Milestone 3

A ton of the refactoring had to do with cleaning up the design of the code. To start, each card object was the same type, even though there were many different card types (Epidemic, City, Special, and Infection). After using the Strategy Pattern to make different Card objects and having an interface which would tie in all the core logic for cards that is shared. This way, we can create, modify, and delete different cards and card behaviors very easily.

We also fixed the Player class the same way. Players used to be all tied to the same player object when they all had different attributes, so we again used better design principles to make different players different objects. All of the same core logic is held together by a player abstract class that all player types (Dispatcher, Gene Splicer, etc.) extend from. This way, we can create, modify, and delete different players and player behaviors very easily.

The rest of the refactorings were to mostly clean up the legibility and flow of the code. We went around looking at if/switch statements with terrible structures, empty classes/methods, excessive comments, duplicate code, feature envy, and data classes and cleaning them up where we could. We resulted in much cleaner code and much more clarity in our code. It is much easier for someone new to read through the project and understand how the project works. It is also much easier to modify the project since we clarified the code.

After all the refactoring, all of our tests still work and our program runs smoothly.

6.2 Milestone 4

6.2.1 Milestone 4 Refactor Analysis

A refactoring of the duplicate code smell of the SetDiseaseCube() method resulted in refactoring of Cures that made it easier to change the cure state. This way, if the game was to be expanded on even more (and more cure states were available), we could easily add more cure states. The long method bad smell present in the infectCity() method of the InfectorBL class was another major refactoring. This method was bloated because it was full of duplicate code and switch statements, both of which were removed for a more concise approach that kept the same logic.

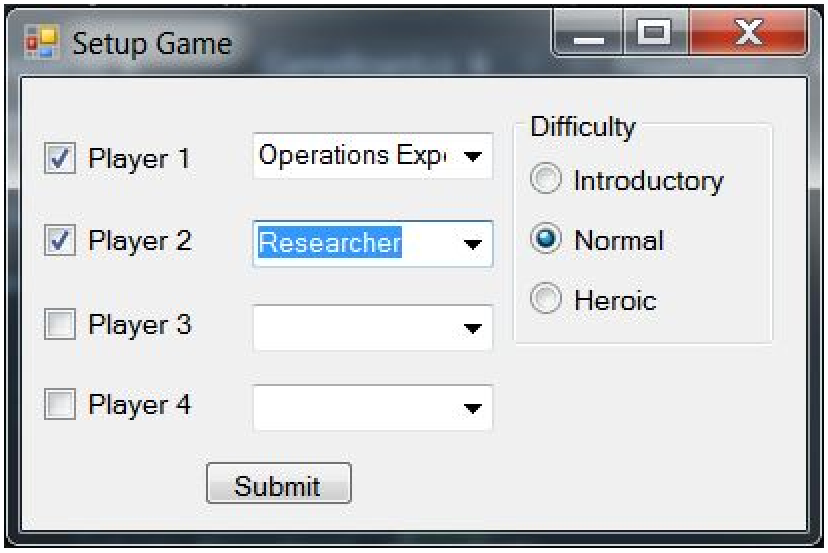
*Pandemic official rulebook: https://goo.gl/CcLs10

User Guide

1. Base Game Rules

The base game rules can be accessed at http://goo.gl/OWZ3Uo. Our game follows all of the rules set out in this document, and does not allow for cheating.

2. Starting the Game

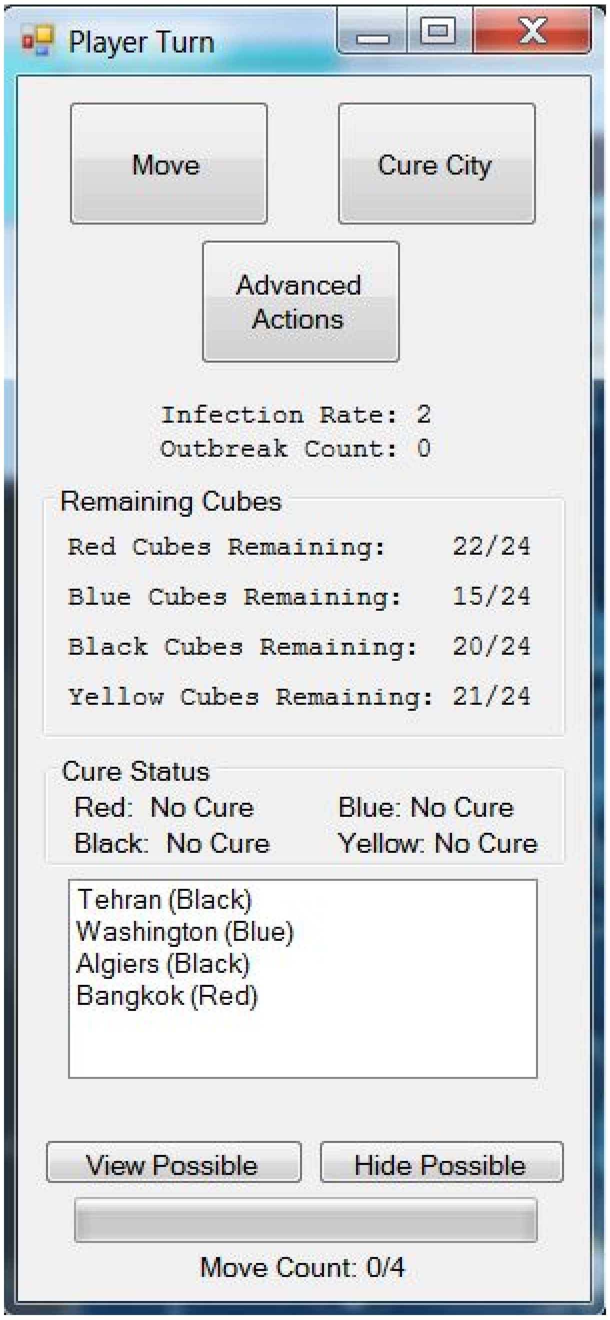

Setup Game

[✓] Player 1	Operations Exp	Difficulty
[✓] Player 2	Researcher	○ Introductory
[ ] Player 3		● Normal
[ ] Player 4		○ Heroic

*Pandemic official rulebook: https://goo.gl/CcLs10

When you load the game, the first form which will show up will allow you to select the number of players you would like to have and their roles, as well as the level of difficulty at which you would like to play the game. To add more players, select the checkbox next to the Player tag. For each player that you select to have play, select a role using the dropdown next to player's name.

3. Performing Player Actions



```
Infection Rate: 2
Outbreak Count: 0
```

Remaining Cubes

```
Red Cubes Remaining:    22/24
Blue Cubes Remaining:   15/24
Black Cubes Remaining:  20/24
Yellow Cubes Remaining: 21/24
```

Cure Status

Red: No Cure	Blue: No Cure
Black: No Cure	Yellow: No Cure

Tehran (Black)
Washington (Blue)
Algiers (Black)
Bangkok (Red)

Move Count: 0/4

Fig. 1

*Pandemic official rulebook: https://goo.gl/CcLs10

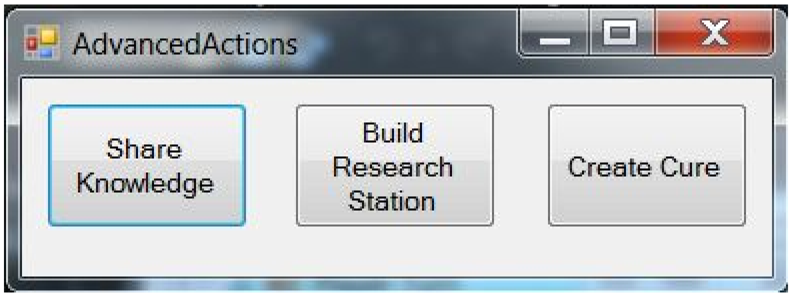


Fig 2.

A. General Actions

3.A.1 Move

In order to move, select move from the player panel (Fig 1) and then the city that you would like to move to on the map. The game will automatically handle any special types of movement possible.

3.A.2 Treat City

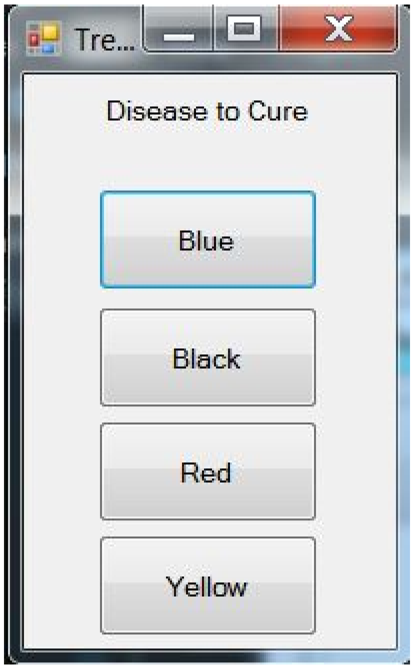


Fig. 4

In order to treat a city, select cure city on the player form (Fig 1) and then the color of the disease you would like to cure from the panel that pops up.

*Pandemic official rulebook: https://goo.gl/CcLs10

3.A.3 Use Card

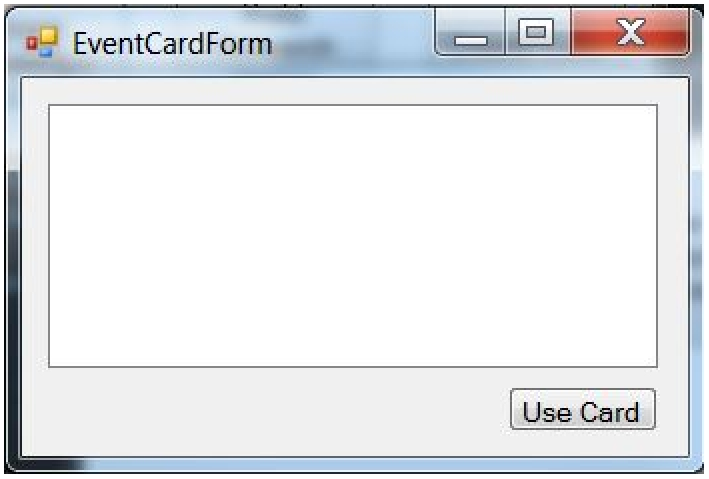


Fig 3.

Select the event card that you would like to use from the form and click Use Card, depending on the card, you will either have a new panel show up or will be able to take an action on the board.

3.A.4 Cure Disease

Select any cards that you would like to use to create a cure, then click submit. The program will automatically check the cards to make sure you can make the cure

B. Specific Role Actions

3.B.1 Dispatcher

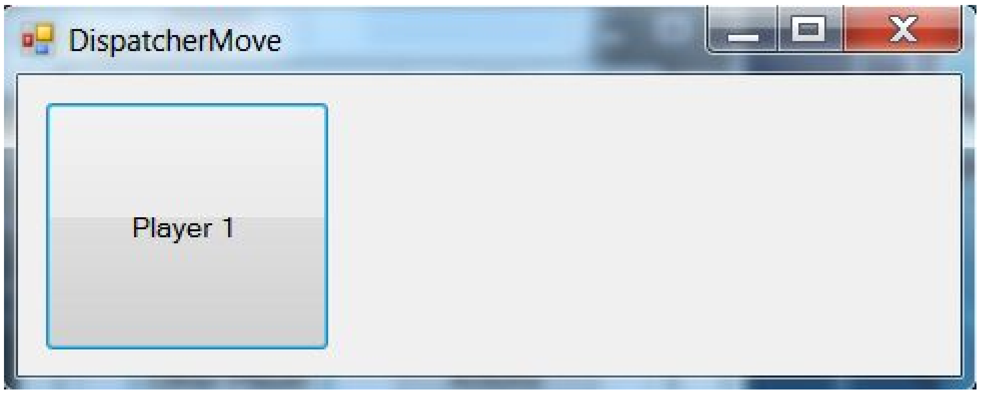


Fig. 5

*Pandemic official rulebook: https://goo.gl/CcLs10

In order to use the dispatcher move, select the button that appears on the Player Panel (Fig 1) and then select the player you would like to control from the panel that shows up (Fig 5) and select the spot on the map you would like to move them to.

3.B.2 Operations Expert

Select the button that appears on the Player Panel to create a Research Station

3.B.3 First Responder

Whenever an epidemic occurs, a message box will show up, at this point, it will be possible for the user to select if the First Responder would like to move to the city where the outbreak is occurring.

Installation and Configuration Guide

Installation:

1. You need a Windows Device (at least Windows 7) and to have .NET 4.5 Framework (or later) installed. If you are missing either of these, please install it before proceeding.
2. Go to https://github.com/cbudo/Pandemic-Windows to get the files for the game. You can either download the files as a .zip or fork the repository.
3. Once you have the files locally on your machine, navigate to it using Windows File Explorer.
4. Navigate to Pandemic-Windows/SQADemicApp/bin/Release.
5. You should have a file named SQADemicApp.exe. Double click to run the game.
6. Enjoy!

Note: If the game does not run or you are missing files, please try re-pulling the files from the Github repository and repeating the steps above.

Configuration Guide:

Minimum Requirements:

1. Windows 7 (or later)
2. .NET 4.5 Framework (or later)
3. 500MB RAM
4. Processor 2.0 GHz
5. 100 MB free space HDD/SSD

Game Configuration:

1. When you run the game, you should see a screen with player selection. You can play with 2-4 players. Starting with 0 or 1 players will crash the game. The game does not support more than 4 players.
2. Each player must pick a unique role. If two or more players pick the same role, the game will not start.

*Pandemic official rulebook: https://goo.gl/CcLs10

Chris Budo, Andrew Ma, Austin Fahsl, Gene Logan
Milestone 4

Maintenance Guide

Updating

The Pandemic game is packaged within one executable file. There is no upkeep for any individual release of this piece of software. You can download the newest release of this software from https://github.com/cbudo/Pandemic-Windows/releases.

Troubleshooting

Troubleshooting the software involves a simple restart of the program. **Game progress will be lost if this is done.**

Bug Reporting

Report all bugs to any of the following email addresses:
budocf@rose-hulman.edu
fahslaj@rose-hulman.edu
loganga@rose-hulman.edu
maas@rose-hulman.edu

*Pandemic official rulebook: https://goo.gl/CcLs10

Chris Budo, Andrew Ma, Austin Fahsl, Gene Logan
Milestone 4

Software Requirements Specification (SRS)

Overview

Pandemic sets out to provide a collaborative gaming experience that accurately simulates the real board game of the same name. Due to this basis for the application, most of the requirements are simply outlined by the features of the actual game and the GUI necessary to tie these elements together. The following section titled Base Requirements is a detail of those specific requirements. The second section, titled Extended Requirements, will outline requirements that we have added based on CSSE 375 assignments for feature additions.

Base Requirements

Features

Several of the following features were outlined by the previous design document for this project as it was previously a project in CSSE 376, but may have been combined, removed, or edited slightly to better convey features in a more concise manner.

1. **Comprehensive GUI** - The basis of the user interaction will be a Graphical User Interface that combines that displays the board of the game, a panel that represents each player, buttons that allow the player to interact with the board, representations of physical game items such as cards as lists, and configuration windows.

2. **2-4 Player Cooperation** - The actions that players make in the game will work towards the common goal of eradicating all infections by developing cures. All player actions will reflect this (curing, creating research stations, role-specific actions, etc). More detailed explanations of the possible actions and constraints for player interaction are detailed in the official Pandemic rulebook.

*Pandemic official rulebook: https://goo.gl/CcLs10

Chris Budo, Andrew Ma, Austin Fahsl, Gene Logan
Milestone 4

Extended Requirements

Game Logic Changes

The changes in this section involve modifying the logic of the game from its previous requirements specification. Due to the scope of this class and the project length, implementing the entire State of Emergency (SoE) expansion pack was not feasible. Instead, some major, complex changes were chosen to be added.

1. **Difficulty Settings -** Despite being a feature in the real game, the previous requirements for this project chose not to implement different levels of difficulty. Here they will be implemented as an Easy, Medium, and Hard difficulty.

2. **New Player Roles -** The Gene Splicer, First Responder, and Pharmacist player roles from the SoE expansion will be added into the game with all of their role-specific behavior implemented. Other roles from the expansion were excluded because of their inability to be added without adding in the rest of the expansion.

Quality of Life Enhancements

These changes were chosen based on perceived lack of understanding of the original implementation of the GUI. They do not add any new logic or rules to the game, but simply allow the player to understand information about the game in a clearer way.

1. **Tooltips for Card and Cities -** Important information, such as card descriptions and the number of infection cubes currently at a given city, were not accessible in the previous iteration of this project. These will be implemented as clickable tooltips.

2. **Help Panel for the Setup Window -** There were previously no instructions for how to use the startup configuration options. This will be implemented text that can be seen by hovering over options in the startup menu.

3. **Show Highest Possible Moves -** The GUI will offer a count of how many possible moves can be made based on the current circumstances.

*Pandemic official rulebook: https://goo.gl/CcLs10

4. **Show Remaining Card Count for Decks -** Since the game can end when the deck runs out, this was determined as important information and will be implemented on the Draw button.

*Pandemic official rulebook: https://goo.gl/CcLs10

Chris Budo, Andrew Ma, Austin Fahsl, Gene Logan
Milestone 4

Software Architecture Design Specification (SADS)

Overview of Previous Architecture

The design of Pandemic was largely in the hands of the previous iterations' contributors. There was no previous documentation on the architecture design of the project since the class it was written for likely did not require it. Coincidentally, the actual architecture of the project reflects this lack of foresight as much of the previous design was seemingly haphazard. Most fields and methods were made public and static and called wherever they were needed without any regard for object oriented design. The code overall was very highly coupled. However, a basic structure still exists, and the main classes that handle the bulk of the work are GameBoardModels, Create, Player, and the classes in the BL (board logic) package. GameBoardModels encapsulates many of the objects that represent items such as cards and cubes. Create initializes most of those encapsulated objects for GameBoardModels. The Player class holds all the information for how a player should perform their role-specific actions. Finally, the classes in BL contain loosely cohesive methods that deal with board actions like infecting cities.

Changes to Architecture

Tackling every single design issue present within the Pandemic project was simply not a feasible task for the class's time constraints. Instead, much of the effort was focused on key classes in the project such as Player and Card. These classes were turned into polymorphic structures that allow for easier extension when adding features. Additionally, new classes were added that encapsulate important structures and behavior such as the Deck class which not only removes helper classes but provides a more object-oriented data structure for managing Cards. Overall, the interactive structure of the major classes like GameBoardModels and Create had to stay the same due to the overly high level of coupling, but smaller internal changes allowed for a much better, more object-oriented design.

*Pandemic official rulebook: https://goo.gl/CcLs10

Test Suite

Unit Tests

The first important tests that need to be implemented first before anything else is Unit Tests. Unit tests ensure that our code work on the most basic level, so that when it comes time to integrate different parts of our code, we know that they all work individually. All of our unit tests passed, which means that we can move on to the next phase of testing.

Integration Tests

Once unit tests are complete, we needed to test how each individual component worked with another. These tests ensure that our code not only works in individual components, but as combined parts as well. For example, we tested that a dispatcher could move another player, a city could be infected correctly, a city could be cured correctly from a player move, and so on so forth. All of our integration tests passed, which means that the last test to do was to make sure that our game worked from start to finish.

Redesigning Existing Tests

Because of our refactorings, a lot of tests that came with the project broke. Throughout each milestone, we had to make sure that the tests constantly stayed up to date with the latest project revisions. We also had to create some new tests in order to thoroughly check the correctness of our refactorings.

Since our refactorings only touched bad code smells, our rewritten tests were clearer and better. This makes sense, considering that the old test used and tested ugly code. We also knew that our refactored code would replace break the old code's tests, so we designed the refactorings to improve testing.