**Software Engineering**

* **Term Poject : 윷놀이 게임 -**



**2019.06.08**

**20174507김다빈**

**20174470전희수**

**20161344허정우**

**20163453현도연**

* **목차 및 작성내용**

|  |  |
| --- | --- |
| **목차** | **작성 내용** |
| 1. 팀원 및 역할 분배 | 1. 팀 구성 2. 역할 분배 |
| 1. 비전 | 1. Introduction to the system 2. Main features |
| 1. Use Case Model | 1. Use case Diagram 2. Use case Descriptions 3. System Sequence Diagram 4. Operation Contracts |
| 1. 설계 및 구현 | 1. 설계 2. 구현 3. 보완점 |
| 1. 사용법 및 테스트 결과 | 1. 사용법 2. 테스트 결과(스크린샷) |
| 1. Project Management | 1. github repository주소 2. Progress History |

1. **팀원 및 역할 분배**
2. **팀 구성**

해당 팀은 김다빈, 전희수, 허정우, 현도연 4명의 팀원으로 구성되어 있다.

1. **역할 분배**

김다빈 –

전희수 –

허정우 –

현도연 -

1. **비전**
2. **Introduction to the System**

윷놀이 게임이란 ‘윷판’과 4개의 ‘윷’을 이용한 일종의 보드게임으로 한국의 전통놀이이다. 4개의 ‘윷’을 던지면 던져진 모양에 따라 ‘도,개,걸,윷,모,빽도’와 같이 총 5가지의 결과가 나오는데 이에 따라 말을 움직이며 한 팀에 배정된 모든 말을 도착지점까지 도달시키면 승자가 된다. 사용자는 본 시스템을 이용하여 ‘윷판’과 ‘윷’이 없더라도 윷놀이 게임을 즐길 수 있다.

1. **Main Features**

* 본 시스템은 Qt 라이브러리를 이용해 제작된다.
* 사용자는 참여자의 인원수(2~4)와 게임 말 개수(2~5)를 슬라이드바를 이용해 조절하여 선택할 수 있다.
* 일정 순서에 따라 참여자는 돌아가면서 윷을 던질 수 있다.
* 참여자는 랜덤 혹은 지정버튼을 클릭하여 윷의 결과를 얻는다.
* ‘도개걸윷모’ 순서대로 사용자는 1,2,3,4,5칸씩 이동할 수 있다.
* 사용자는 자신이 움직이려는 말을 클릭하면 그 말이 이동할 수 있는 위치가 표시되고 그 중에 하나를 선택하여 말을 움직인다.
* ‘윷’또는 ‘모’가 나올 경우 윷을 한번 더 던질 수 있는데 이때 윷을 던져서 나온 순서대로 말을 옮겨야 한다.
* 말이 ‘윷판’의 꼭지점(그림 넣고 해당 위치 표시 등으로 수정)에 정확히 위치했을 때만 갈림길로 들어갈 수 있는 기회가 생긴다.
* ‘윷판’의 꼭지점(그림 넣고 해당 위치 표시 등으로 수정) 위치했을 때 ‘빽도’가 나올 경우엔 진행방향의 반대방향으로 가되 원하는 길로 갈 수 있다.
* 같은 칸에 같은 팀의 말이 동시에 위치하게 되면 말을 업을 수 있고 다른 팀의 말이 있으면 다른 말을 잡을 수 있다.
* 참여자 중에 가장 먼저 모든 말을 통과시키는 사람이 우승을 하고 승자를 표시하는 화면이 나타난다.
* 도, 빽도, 빽도를 제외한 나머지 순서로 윷을 던지면 말을 골인지점으로 이동할 수 있다.

1. **Use Case Model**
2. **Use Case Diagram**
3. **Use Case Description**

* Scope : 윷놀이 application
* Level : user goal
* Primary Actor – Player
* Stakeholders and Interests
* Preconditions: Player is identified and authenticated.
* Success Guarantee(or Postconditions)
* 사용자는 해당 판의 결과(승자)를 알 수 있다.
* 윷을 던지면 해당 수만큼 말이 정확히 이동한다.
* 한 플레이어의 순서가 지날 때마다 던진 윷의 결과가 바로 보드에 적용된다.
* 한 플레이어의 순서가 지날 때마다 게임이 종료되는지 확인된다.
* 랜덤 윷을 선택한 경우 랜덤하게 윷의 결과가 나온다.
* Main Success Scenario

1.     플레이어가 총 몇 명의 플레이어와 참여할지, 한 팀당 몇 개의 말로 진행할지 결정한다.

2.     게임을 시작(시작 버튼을 클릭)한다.

3.     플레이어는 자기 차례에 랜덤 윷 버튼(?)을 클릭해 윷을 던진다.

4. 이동할 수 있는 말 중 플레이어가 이동할 말을 클릭한다.

5.     클릭한 말이 이동 가능한 위치가 보드에 하이라이팅 된다.

6.     하이라이팅 된 위치 중 이동할 위치를 클릭한다.

7.     말이 해당 위치로 이동하여 보드에 나타난다.

3-7의 과정을 게임이 종료될 때까지 반복한다.

8.     한 플레이어의 모든 말이 먼저 다 돌아온 경우(모든 말을 보드 밖으로 뺀 경우) 해당 플레이어가 승리한다.

9.     승리한 플레이어의 정보를 팝업창으로 띄운다.

10.     팝업창에서 종료 버튼을 누르면 게임을 종료한다.

* Extensions (or Alternative Flows):

**\*a. 윷을 던질 때 자신이 원하는 윷의 결과를 지정하여 윷을 던질 수 있다**

  1. 지정 윷 버튼을 클릭한다.

2. 도 개 걸 윷 모 빽도 중 원하는 버튼을 클릭하여 던진다.

3. 지정 윷의 결과에 따라 해당 팀의 말이 이동 가능한 위치가 보드에 하이라이팅된다.

4. 하이라이팅 된 위치 중 이동할 위치를 클릭한다.

5. 말이 해당 위치로 이동하여 보드에 나타난다.

**\*b. 언제라도 시스템을 종료할 수 있다.**

임의로 시스템을 종료할 시에 해당 게임의 진행 상황을 저장하지 않고 종료된다.

해당 진행상황을 복구할 방법은 없다.

**\*c. 모든 플레이어는 해당 차례에 이동 가능한 위치에 자신의 다른 말이 있을 때 업을 수 있다.**

1. 플레이어가 버튼을 클릭해 윷을 던진다.
2. 해당 말이 이동 가능한 위치가 하이라이팅 된다.
3. 하이라이팅된 위치 중 같은 팀의 말이 존재하는 경우, 그 위치를 클릭하여 말을 업는다.
4. 플레이어의 말 두 개가 보드의 같은 위치에 나타나고 이후 함께 이동한다.

**\*d. 이동 가능한 위치에 상대편의 말이 있을 때 상대편의 말을 잡을 수 있다.**

1. 플레이어가 윷을 던진다.
2. 이동 가능한 위치가 하이라이팅된다.
3. 하이라이팅 된 위치 중 상대편의 말이 있는 경우, 플레이어가 상대편의 말이 있는 곳으로 이동한다.
4. 해당 위치에 있던 상대편의 말은 플레이어의 말에게 잡힌다.
5. 상대편의 말이 출발점으로 되돌아가고 해당 말은 다시 출발점에서 시작해야 한다.
6. 플레이어는 윷을 또 던진다.

**\*e. 이동 불가능한 위치(하이라이팅 되어 있지 않은 위치)를 클릭할 수 없다.**

1. 플레이어가 윷을 던진다.
2. 이동 가능한 위치가 하이라이팅 된다.
3. 하이라이팅되지 않은 위치를 클릭한다.
4. 변화가 없고 하이라이팅된 위치를 클릭하는 이벤트를 기다린다.

\*f. 모랑 윷 던지면 다시 한다.

1. **System Sequence Diagram**
2. **Operation Contracts**
3. **설계 및 구현**
4. **설계** ---domain model, class diagram -UP다.
5. **구현** ---Sequence Diagram, state chart, mvc 설명, ooad 설명
6. **보완점** --- Qt의 한계로 인한
7. **사용법 및 테스트 결과**
8. **사용법**
9. **테스트 결과**
10. **Project Management**
11. **Github Repository 주소**
12. **Progress History**

* Operation Contracts

precondition – assumptions about the state of the system or objects in the Domain Model before execution of the operation. non-trivial assumptions the reader should be told

postcondition – state of objects in the Domain Model after completion of the operation

**Contract : 1. 팀, 말 개수 정하기**

Operation: 팀말개수정하기(Team: integer, Mal: integer)

Cross References: Use Cases: 팀 수 정하기, 말 수 정하기

Preconditions:

Postconditions:

**Contract : 2. 보드 초기화**

Operation: initializeBoard(Team: integer, Mal: integer)

Cross References: Use Cases: 팀 수 정하기, 말 수 정하기

Preconditions:

Postconditions:

**Contract : 3. 윷 던지기**

Operation: throwYut()

Cross References: Use Cases: 윷 던지기

Preconditions:

Postconditions:

**Contract : 4. 이동할 말 선택**

Operation: chooseMal()

Cross References: Use Cases: 말 이동하기

Preconditions:

Postconditions:

**Contract : 5. 갈 수 있는 곳 표시**

Operation: highlightPossiblePoint(Mal: Mal)

Cross References: Use Cases: 말 이동하기

Preconditions:

Postconditions:

**Contract : 6. 말 이동할 곳 선택**

Operation: setPoint(Mal: Mal, Cell: Cell)

Cross References: Use Cases: 말 이동하기

Preconditions:

Postconditions:

**Contract : 7. 보드 업데이트**

Operation: updateBoard(Board:Board)

Cross References: Use Cases: 말 이동하기

Preconditions:

Postconditions:

**Contract : 8. 승자 표시**

Operation: displayWinner(Team:Team)

Cross References: Use Cases: *게임 종료*

Preconditions:

Postconditions:

**Contract : 9. 게임 종료**

Operation: gameOver()

Cross References: Use Cases: 게임 종료

Preconditions:

Postconditions: