<기능적 요구사항>

1. 선박은 세 칸, 세로로 구성되도록 한다.
2. 선박의 효과적 배치를 위해, 선박의 세 칸 중 중심칸을 우선적으로 배치하고 나머지 두 칸을 배치.
3. 선박의 중심칸의 효율적 배치를 위해, 선박의 중심을 둘 수 있는 행과 열의 좌표 리스트 생성.
4. 선박의 중심이 배치된 지점의 행과 열이 튜플로 묶여서 저장되어야 한다.
5. 선박 중심은 가능한 칸들 중에서 랜덤으로 결정되게 한다.
6. 선박의 중심이 배치되면, 해당 행&열은 다음 선박이 배치될 수 있는 공간에서 제외.
7. 재시작 버튼 생성.
8. 컴퓨터가 타격할 수 있는 행&열을 생성.
9. 컴퓨터가 타격한 지점에 유저의 선박이 있을 경우, 컴퓨터는 해당 타격지점으로부터, 그 지점에 정박중이었던, 유저의 선박 모양을 유추하여, 나머지 부분이 존재할 확률이 높은 지점을 타격하도록 한다. 이 때, 다음 지점으로 당연한 지점은 연달아 타격을 하도록 하고, 확률이 반반인 지점은, 랜덤으로 결정하도록 한다. (AI의 구성)
10. 컴퓨터의 타격순서 결정은 첫 명중 지점의 ‘행’을 기준으로 분류한다. 이 때, 이후 공격해야할 지점의 행이 독특한 파트들은 그 자리에서 처리하되, 이후 공격해야할 지점의 행이 규칙적으로 나타나는 경우들은 묶어서, 인자들을 가지는 메소드로 구현하도록 한다.

2. 사용자 인터페이스 요구사항

2.1. 게임의 사용자 인터페이스 구성요소

2.1.1. 게임 창 제목에 게임 이름을 넣을 것 : BATTLESHIP GAME

2.1.2 맨 위 쪽에 게임 이름을 명시할 것 : <WELCOME TO BATTLESHIP GAME>

2.1.3 게임을 위한 10x10 칸의 필드 구성을 컴퓨터 쪽 필드는 좌측에, 유저 쪽 필드는 우측에 위치시킬 것.

2.1.4 게임 진행을 안내할 안내 메시지 출력 창을 컴퓨터 쪽 필드 아래에 위치시킬 것.

2.1.5 게임 재시작을 위한 버튼을 사용자 쪽 필드 아래에 위치시킬 것.

2.1.6 두 필드를 직관적으로 구분할 수 있도록 구분되게 할 것.

-> 날짜 위젯 추가로 구분

2.1.7 게임 창 크기가 상당히 크므로, 화면 상단에 출력되도록 위치를 조정할 것.

3. 소프트웨어 구조 설계

단일모듈+ : Battleship.py

클래스 : Battleship

메서드 : \_\_init\_\_

역할 : 게임이 시작될 때 마다 새롭게 UI를 구성. 제목과 컴퓨터 쪽 필드와 유저 쪽 필드, 게임 안내 메시지 창과 날짜 위젯, 재시작 버튼을 구성. 그리고 유저의 배를 랜덤하게 3척을 뽑아 유저 쪽 필드에 배치시킨 뒤 유저가 알 수 있도록 유저 쪽 필드에 명시하도록 하고, 내부 리스트에 저장. 컴퓨터 쪽의 배도 랜덤하게 3척을 뽑지만 유저가 알 수 없도록 전부 동일한 문자가 새겨진 버튼으로 배의 위치를 숨겨놓고, 프로그램 내부 리스트에 저장해 둠.

메서드 : OnCenter()

역할 : 게임 창의 크기가 상당히 크므로 게임 시작 혹은 재시작 시에 게임이 사용자 화면의 상단에서 시작될 수 있도록 하는 메서드.

메서드 : Remove\_Dup\_user

역할 : 앞서서, 선박배치가 끝나게 되면 그 다음 배가 절대 배치될 수 없는 좌표들이 존재한다. 열은 고정되어 있으니, 불가능한 행들을 제거하는 과정을 거쳐야 한다. 이 부분들을 선박 배치 후보 좌표들에서 빼줘야 한다. 해당 과정을 담당하는 메서드이다.

메서드 :Remove\_Dup\_ai

역할 : 앞선 Remove\_Dup\_user와 동일한 목적이다. 선박 배치의 대상이 ai냐 user냐의 차이가 존재한다.

메서드 : AIButtonClicked

역할 : Ai의 해역에 공격이 가해질 시, 공격이 빗나감,명중함에 따라서, 화면상에 처리되는 것을 다르게 함.

메서드 : Hit (1,2,3이 존재)

역할 : 히트는 ai가 타격한 좌표에 실제 user의 선박이 존재할 경우, 발동되는 메소드로, 유저의 선박 좌표에서, 타격 대상으로 지정된 좌표를 제외시켜주는 메소드가 된다. 추가적으로 파괴를 의미하는 버튼의 작동에도 기여한다.

메서드 : Gameplay

역할 : 게임 재시작에 대응되는 메서드이다.

메서드 : ButtonClicked

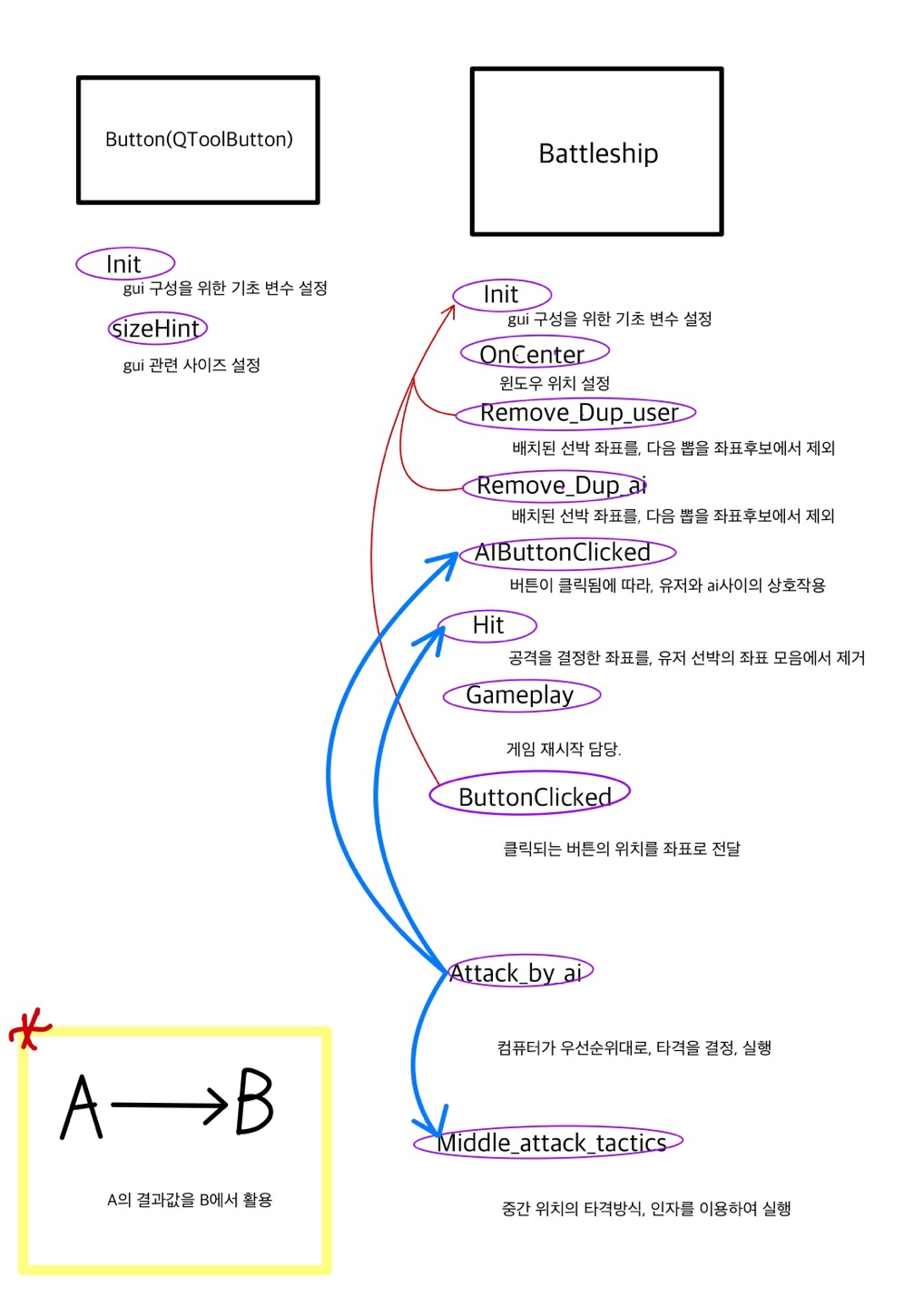
역할 : 없음. 사용자 필드의 버튼을 연결해 두기만 하는 메소드.

메서드 : Attack\_by\_ai

역할 : 공격 후보 리스트에서, 튜플을 하나 랜덤으로 뽑아서 타격을 시작한다. Ai는 user의 선박이 어디 있는지에 관련한 정보가 아예 없는 상황에서는, 랜덤으로 접근을 시작하고, 그렇게 뽑은 것이 명중을 했을 때부터, 본격적으로 명중한 배의 위치를 추정하는 과정이 시작되게 구현함.

메서드 : Middle\_attack\_tactics

역할 : 컴퓨터가 유저를 공격할 때 사용하는 메소드. 4-4의 4번째 문단에서 자세히 설명.



4. 구현 상세 설계

4-1. 레이아웃 설정

self.setWindowTitle : 게임 창이 열렸을 때 탭에 표시되는 이름

self.totalLayout : 전체 레이아웃 구성을 위한 틀 (QgridLayout)

self.titleLabel : 게임 창 맨 위에 표시될 게임 이름

self.titleGrid : self.titleLabel을 포함하여 self.totalLayout에 추가하게 될 레이아웃

self.gridAI:컴퓨터 쪽 필드를 나타낼 그리드 형태의 레이아웃, addWidget 후self.totalLayout에 추가

self.gridAI.addWidget : self.gridAI에 버튼을 추가하는 역할

self.messageLayout : 게임 안내 메시지 부분 추가를 위한 레이아웃, self.totalLayout에 추가 될 것

self.messageLine : 게임 안내 메시지를 출력할 라인. self.messageLayout에 추가 할 것, 읽기 전용

self.gridUser:유저 쪽 필드를 나타낼 그리드 형태의 레이아웃, addWidget후self.totalLayout에 추가

self.gridUser.addWidget : self.gridUser에 버튼을 추가하는 역할

self.date : 날짜 위젯

self.dateLayout : self.date를 추가하여 self.totalLayout에 추가 될 레이아웃

self.restartButton : 재시작 버튼

self.restartLayout : self.restartButton 을 추가하여 self.totalLayout에 추가 될 레이아웃

def onCenter(self) : 게임 창이 크므로 사용자 화면 상단 쪽에서 실행되도록 하는 역할

def Gameplay(self): 현재 열려 있는 객체의 실행을 중단하고 새로운 객체 생성 후 실행

4-2 랜덤한 배를 설정하는 Logic

Self.hori : 0부터 9까지 담긴 리스트

Self.verti : 1부터 8까지 담긴 리스트

self.user\_selected : 유저의 선박 중심이 배치될 좌표들을 포함한 리스트이다. 선박 세 칸중 중심이 이 위치할 좌표가 이 리스트 안에 들어가 있다.

self.ai\_selected : self.user\_selected와 비슷한 맥락이다. 선박의 주인이 ai라는 점이 차이가 된다.

self.attacked\_location : ai의 공격 대상 좌표들로 구성된 리스트가 된다.

0부터 9까지 담긴 리스트 self.hori와 1부터 8까지 담긴 리스트 self.verti 생성, 이를 이용한 이중 for문으로 필드에서 뽑아낼 배의 중앙값을 튜플의 형태로 선택한다(중앙값 선택 후 위 아래로 선택할 것). Random 모듈로 리스트에서 임의의 튜플을 뽑아내고 뽑아낸 값의 위 아래를 배로 설정한다. 이를 3번 반복하는데, 중복해서 배를 선택하지 않게 하기 위한 remove\_Dup\_ai, remove\_Dup\_user 메소드를 작성하여 중복 없이 선택하도록 한다. remove\_Dup\_ai와 remove\_Dup\_user는 둘 다 중앙값에서 위 아래로 두 칸에 해당되는 튜플을 지워서 다시 뽑을 수 없게 하는 메소드이다. 컴퓨터 쪽 필드와 사용자 쪽 필드에서 배를 뽑는 로직 자체는 동일하지만, 사용자 쪽 필드에서 배를 뽑는 경우에는 self.user\_selected 리스트에 좌표를 저장하고 사용자가 자신의 배 위치를 확인할 수 있도록 배 위치의 버튼을 바꿔줘야 하고, 컴퓨터 쪽 필드에서 배를 뽑는 경우에는 사용자가 컴퓨터 쪽 필드의 배 위치를 알 수 없도록 버튼을 건드리지 않고 self.ai\_selected 리스트에 좌표를 저장만 해둔다.

4-3 사용자가 컴퓨터를 공격

사용자는 컴퓨터 필드의 버튼을 하나씩 클릭하여 공격을 시도한다. 사용자가 컴퓨터 필드의 버튼을 클릭하게 되면 이 버튼은 AIButtonClicked 메소드로 연결된다. AIButtonClicked가 작동하는 방식은 클릭 받은 버튼의 좌표를 구한 뒤, 이 좌표가 self.ai\_selected 리스트 안에 있을 경우 배 공격에 성공했음을 알리는 메시지를 출력하고 별 모양 버튼으로 바뀌며 비활성화된다. 또한 해당 좌표는 self.ai\_selected에서 지워진다. 그렇지 않을 경우 공격에 실패했음을 알리는 메시지를 출력하고 X 모양 버튼으로 바뀌며 비활성화 되며, 컴퓨터가 사용자를 공격하게 된다. 만약 self.ai\_selected의 길이가 0이 되는 순간, 즉 컴퓨터 쪽의 필드의 배를 사용자가 모두 찾아 지워낸 순간, 컴퓨터 쪽의 모든 버튼이 ! 버튼으로 바뀌며 비활성화 되며 승리를 알리는 메시지를 출력한다.

4-4 컴퓨터가 사용자를 공격

컴퓨터는 사용자 쪽 필드의 한 좌표를 랜덤하게 self.attacked\_location 에서 뽑아낸다. 그 후 사용자 쪽 필드의 좌표와 비교한 뒤 배의 좌표가 아니라면 뽑기를 그만둔다.

만약 컴퓨터가 타격에 성공했을 경우에는 크게 3가지 경우가 존재한다. 2,9번째 행의 좌표를 타격한 경우, 1,10번째 행의 좌표를 타격한 경우와 그 외의 경우이다.

2번째 행의 좌표를 타격한 경우에는 배가 존재할 수 있는 경우는 1,2,3 번째 행에 위치 혹은 2,3,4 행에 위치로 두가지 경우가 있다. 만약 이렇게 된다면 3번째 행에는 반드시 배가 위치하게 된다. 그렇기 때문에 3번째 행의 좌표를 즉시 공격하고, 그 후 랜덤하게 1번째 행 혹은 4번째 행의 좌표를 공격한다. 8번째 행의 좌표를 타격한 경우에도 같은 논리로 무조건 7번째 행에는 배가 위치하므로 7번째 행의 좌표를 즉시 공격하고, 6번째 혹은 9번째 행을 랜덤하게 선택하여 공격한다.

1번째 행의 좌표를 타격한 경우에는 배가 존재할 수 있는 경우가 1,2,3번째 행에 위치하는 단 하나 뿐이다. 따라서 즉시 3개의 배 좌표를 공격한다. 10번째 행의 좌표를 타격한 경우에도 같은 논리로 8,9,10 번째 행에 위치하는 좌표를 공격한다.

그 외의 경우에는 타격에 성공한 값 위 아래로 한 칸을 랜덤하게 선택하여 공격을 시도한다. 이 공격은 middle\_attack\_tactics 메소드에서 정의된다. 랜덤하게 선택하여 시도한 공격이 성공한다면 그 공격 위의 좌표 혹은 첫 타격에 성공한 값 아래를 공격한다. 예를 들어, 5번째 행의 좌표를 타격 성공했을 때, 랜덤하게 4번째 혹은 6번째 행의 좌표를 타격 시도한다. 4번째 행의 좌표를 타격 시도했을 때, 그 시도 또한 성공한다면 3번째 혹은 6번째 행의 좌표를 타격 시도하는 것이다. 만약 이 과정 중에 타격이 실패 할 경우, 실패 한 시점의 좌표를 self.save 리스트에 저장해두고 다음에 컴퓨터가 사용자를 공격할 때 self.save에 저장된 좌표부터 타격한다.

어떤 경우든, 타격에 성공시 self.user\_selected에 있는 좌표가 지워진다. 만약 self.user\_selected 의 길이가 0이 될 때, 즉 유저의 배가 모두 공격 당했을 때 컴퓨터 쪽의 필드의 버튼이 모두 ! 버튼으로 바뀌며 패배를 알리는 메시지를 출력한다.

5. 테스트 보고

|  |  |  |  |
| --- | --- | --- | --- |
| 테스트 | 테스트 상황 | 설명 | 성공 여부 |
| 1 | 게임 실행 및  재시작 | 정상적으로 코드가 실행되며 창이 생성되고,  재시작 버튼이 제대로 작동하는지 | PASS |
| 2 | 유저가 컴퓨터  공격 성공 | 유저가 컴퓨터 필드의 배를 찾았을 때, 의도한 이벤트로 연결되는지 | PASS |
| 3 | 유저가 컴퓨터  공격 실패 | 유저가 컴퓨터 필드의 배를 찾지 못했을 때, 의도한 이벤트로 연결되는지 | PASS |
| 4 | 컴퓨터가 유저  공격 성공 | 컴퓨터가 유저의 배를 찾았을 때,  의도한 이벤트로 연결되는지 | NON-PASS |
| 5 | 컴퓨터가 유저  공격 실패 | 컴퓨터가 유저의 배를 찾지 못했을 때,  의도한 이벤트로 연결되는지 | PASS |
| 6 | 이벤트에 따른  메시지 출력 | 이벤트에 연결되었을 때,  그에 맞는 메시지가 출력되는지 | PASS |
| 7 | 유저 패배 | 유저의 배가 모두 공격당했을 때,  의도한 이벤트로 연결되는지 | PASS |
| 8 | 유저 승리 | 컴퓨터의 배가 모두 공격당했을 때,  의도한 이벤트로 연결되는지 | PASS |

6. 보완할 점

첫 번째로, 현재 코드에서 그리드 레이아웃 상의 버튼을 호출하였을 때,

그 버튼이 그리드 레이아웃 상의 몇 행 몇 열에 위치하는지 정확하게 알 수 없음. 지금은 버튼의 x, y position을 알아내어 적절한 값을 대입하여 계산해 좌표를 찾아냈지만, ‘적절한 값’이 사용자마다 다르다는 것을 확인했음. 이로 인해 게임을 플레이하는 사용자마다 적절한 값을 찾기 위한 보정 작업이 필요해짐. 즉 사용자에 따라 코드가 다르게 실행됨. 따라서 모든 사용자에게 동일한 경험을 보장할 수 없기 때문에 보완해야 할 부분임.

두 번째로는, 경우를 나눌 때 발생했는데,

코드상에서 ai가 명중을 하였을 시, 다음 타격지점을 찾는 로직을 구성할 때, ai가 명중한 지점의 바로 위와 아래 칸이 아직 공격이 시도되지 않은 칸이란 가정하에 해당 코드가 적혀 있는데, 실제로 반례를 들어본다면, 컴퓨터가 5행 4열을 타격 시 아무것도 없었다는 판정 이후에, 4행 4열을 타격하여, 유저의 배에 명중하는 경우가 생길 경우, 명중한 칸 아래인 5행 5열이 이미 배가 없는 지점이란 것이 밝혀진 상태가 되어, 의도된 부분대로 ai의 공격이 작동되지 않는 다는 것을 알게 되었음. 해당 부분을 후반부에 테스트 케이스를 여럿 넣어보는 과정에서 발견이 되었는데, 해당 로직이 ai의 기본 작동원리의 너무 큰 부분을 차지하고 있어서 변경이 불가능 하였음. 이를 통하여, 초반에 로직의 경우를 나누는 부분에서 충분히 모든 경우에 대한 고민이 되어야 했음을 되짚어 볼 수 있었고, 정상적으로 작동하는 ai를 가장 기초적인 수준으로나마 구현하는 것 역시 고려할 사항이 매우 많다는 것을 알 수 있었음.