<프로젝트 전반>

1. 서버는 의존 라이브러리(JDBC)가 있기에 메이븐으로 프로젝트 관리
2. 소켓통신관련 클래스들은 싱글톤으로 정의했는데 다중 스레드환경 동기화 오버헤드를 줄이기 위해 Lazy Holder방식으로 싱글톤 적용
3. 클라이언트의 회원관리쪽에 Callback방식으로 개발
4. 데이터베이스 측 퍼포먼스 향상을 위한 Connection Pool 사용 (라이브러리 사용없이 직접 개발)
5. 소켓을 사용할 때 다중사용자 환경을 위해 스레드 사용
6. 모든 기능을 개발할때 프로그램의 메커니즘을 알기위해 라이브러리에 의존하지 않고 모두 팀원들이 직접 개발함
7. 팀원 역할 분담이 확실하여 각 소스를 모아 합치는데에 반나절도 안걸림
8. 팀원 역할 분담이 확실하여 일주일에 딱 플젝시간 세시간밖에 안모였어도 개발이 수월하게 진행됨

<GUI 및 클라이언트 - 모설아>

1. 바둑돌을 놓을 때 단순히 교차점에 마우스 클릭을 인식하게 하면 구현은 아주 간단하다. 하지만 이런 경우 사용자가 정확히 교차점을 눌러야만 바둑돌이 놓아지기에 아주 불편하다. 때문에 상하 좌우로 일정 범위 안에 들어올 경우 좌표값을 평균 내어 가장 가까운 교차점에 돌이 놓아지도록 마우스 히트박스를 만들었다.
   1. 최초 구현시의 원리 : 모든 교차점들의 좌표를 1차원 배열에 저장하여 사용자 클릭시 좌표평면에서 점사이의 거리를 구하는 Math.sqrt((x2-x1)\*(y2-y1)) 를 통해 이 값이 바둑돌의 반지름과 같은 좌표가있으면 해당 좌표를 리턴하는 방식이었다.
      1. 하지만 이 방식의 가장 큰 문제점은 구현은 간단하지만 이 체크 방식이 너무 비효율 적이라는 데에 있다.
   2. 이후 변경된 방식 : 모든 교차점의 좌표를 전부 체크해볼 필요 없이 계산을 통해 가장 가까이 있을 교차점을 반환.
      1. 먼저 클릭을 통해 전달 받은 좌표에서 바둑판 가로선(세로선에도 같은 방식으로 계산을 하여 y좌표를 도출한다. 식은 같기때문에 x좌료 구하는 방법만 설명하도록 함.)의 시작점을 뺀다.
      2. i 번의 값을 줄간격으로 나눈 몫과 나머지를 계산하여 저장해둔다.
      3. 그리고 나머지 값에 따라 나머지가 줄간격의 절반보다 작으면 몫을 그대로, 절반보다 크면 몫에 1을 더하여 리턴한다. 이 좌표는 정확히는 좌표평면에서의 실제 좌표가 아닌 몇번째 칸인지를 나타내는 로직 좌표이다.
      4. 원리 : 입력 좌표에서 줄의 시작 좌표를 뺀 뒤 줄간격으로 나누게 되면 몫이 바로 각 줄의 논리적 좌표가 된다. 바둑판의 첫 줄이 100부터 그려져있고, 줄간격이 20이라면, 100은 0, 120은 1, 140은 2번째 논리 좌표가 된다. 만약 200이라는 좌표를 받았다고 하면 200-100을 한 뒤 20으로 나누면 값은 5가 나온다. 200은 5번째 논리 좌표가 맞다. 이러한 논리를 사용했을 때, 나머지를 구하고, 나머지의 값이 줄 간격의 절반보다 작으면 아직 이전좌표보다 가깝다고 할 수 있고, 더 크면 이후 좌표에 더 가깝다고 할 수 있다. 이는 반올림의 원리를 이용한 것인데, 207이라는 좌표가 들어왔다면, 이 좌표는 220보다는 200에 가깝다. 이는 논리좌표 5에 더 가까운것이 되므로 (200-100)/20의 몫에 더 가까워지게 되는 것이다.
      5. 어째튼 해당 원리를 가로축과 세로축에 각각 적용하여 x, y 의 논리 좌표를 구할 수 있고, 여기에 시작점+(좌표\*줄간격)을 하면 논리 좌표의 실제 좌표를 구할 수 있다.
      6. 해당 방식은 빠르고 정확하며, 복잡도를 O(n)에서 O(1)로 줄인 획기적인 방식의 알고리즘이라고 생각한다.
   3. a 방식과 b 방식 모두 직접 고안하여 만든 알고리즘이며 어떠한 자료도 참고하지 않았습니다.
2. 사용자가 자신의 바둑돌이 놓아질 위치를 미리보기 할 수 있도록 반투명한 미리보기 그래픽을 구현함.
   1. 해당 위치에 먼저 놓여져있는 바둑돌이 없는지 체크한다.
   2. 빈칸이라면 해당 위치가 의미 있는 위치인지(바둑돌을 놓을 수 있는 좌표평면 안쪽에 있는지) 확인 한 후, 미리보기를 보여준다.
   3. 게임 시작 전에도 미리보기가 떴는데, 이는 필요도 없고 오히려 혼란을 야기할 수 있다고 생각하여, 게임 시작 조건을 체크한 후, 게임이 진행중일때에만 미리보기를 구현하도록 설정함.
   4. 이후에는 자신의 차례일때만, 미리보기가 뜨도록 변경
3. 마우스 포인터 전반에 대한 고민
   1. 구동 테스트 도중, 마우스를 클릭했는데도 불구하고 돌이 놓여지지 않는 버그가 있어서 이 문제가 왜 일어나는 것인가 계속해서 고민했다. 알고보니 이는 사용자가 보통의 컴퓨터 마우스를 사용할 경우, 완벽하게 움직이지 않는 상태에서 클릭을 하는 경우는 많지 않았기 때문이다. 이는 마우스 리스너 클래스가 마우스가 클릭한 상태에서 조금이라도 이동하면 이를 드래그로 인식하기 때문으로 추정되었다. 그래서 바둑돌이 놓아지는 방식을 마우스 리스너가 클릭을 받았을때가 아닌, 클릭이 해제되는 release시에 놓아지도록 변경했더니 이 문제가 해결되었다. 해당 방식은 사용자가 돌을 놓고 싶지 않으면 클릭 해제를 바둑판 밖에 해버리면 되기 때문에 이러한 사용자 경험 역시 만족시킬 수 있게 되었다.
   2. a의 문제를 해결하고 나니 또 발생한 문제가 바로 사용자가 마우스를 클릭했다가 이동하는 경우 바둑돌은 마우스 클릭이 해제된 위치에 정상적으로 놓아지지만, 바둑돌 미리보기는 최초 클릭 위치에 남아서 있는 문제가 있었다. 이 문제 역시 마우스 드래그로 인식하는 경우였기 때문에, Moved 와 Drag양쪽 모두에 미리보기 실행을 넣어둠으로써 해결했다.
   3. 미리보기의 경우 바둑돌을 놓을 수 없는 부분에도 생성되었었는데, 이는 바둑판 라인 안쪽인지 먼저 확인하는 방식으로 해결하였다.
   4. 리스너 부분 알고리즘과 effect부분을 분리해 둠으로써 서버 연결을 위한 수정작업이 용이하도록 하였다.
4. 상단바의 시작 버튼을 마우스를 올려 두었을 때 약간 기울어지게 하여, 생동감 있는 디자인을 구현하였다.
5. 클라이언트 실행시 컴퓨터 자원을 미친듯이 많이 먹는 버그가 있었다. 이는 한동안 해결하지 못했었는데, 긴시간 고민 끝에, 이같은 일은 GameBoardDesign의 repaint메소드가 무한루프를 돌며 딜레이 없이 구성되고 있기 때문이라고 판단했다.
6. 이 문제를 해결하기 위해 처음에는 topdesign에서는 repaint를 아예 없애버렸고, 게임보드의 repaint는 Stone클래스에 넣어, 돌이 놓일때마다 repaint메소드를 호출하는 방식을 사용하였다. 우선 자원 과점 문제는 해결하였지만 이 방식은 MVC구조를 무너뜨리는 구조였고, 오목이라는 특수한 경우를 제외하면 사용할 수 없는 편법에 그쳤다.
7. 그래서 별도의 스레드를 RepaintThread클래스를 통해 일정 시간 간격으로 화면을 repaint해주도록 바꿨더니, 완벽하게 문제가 해결되었다. 현재는 1000/60 즉, 60fps로 구현되어있다.
8. 프로그램의 기본이긴 하지만, 모든 수치값을 메인 클래스에 상수로 구현하여 프로그램의 편의성을 높였다.
9. Stone클래스를 별도로 만들고 해당 객체 배열을 구성하는 식으로 바둑돌을 구현하였으나, 이는 시간상으로도 공간상으로도 상당히 비효율적인 방식이었기 때문에 교수님의 조언에 따라 이차원 배열에 0 1 2의 값으로 각각 비었음, 흑돌, 백돌을 나타내는 방식으로 간단화 하였다.)