결과 보고서

팀2_JVNU_염선욱_201921786

이번 프로젝트 결과 보고서의 주요 내용으로 게임 플레이 방식, 프로젝트의 전체 구조, 주요 코드 리뷰, 프로젝트를 진행하면서 어려웠던 점, 개선해야 할 점 등이 있다. 게임 플레이 방식부터 설명하면, 게임 첫 화면은 대기화면으로, 엔터를 누르면 게임이 시작된다. 게임 컨셉은 마을로 향하는 몬스터가 나오는 차원문을 기사인 주인공이 부셔서 차단하는 것이다. 문이 열려있는 5초 동안 문을 부수지 못하면, 그 문에서 몬스터가 한 마리 나온다. 모든 몬스터는 마을이 있는 맵 왼쪽으로 향한다. 몬스터가 마을로 향하면 주인공의 목숨이 한 개 줄어든다. 60초 동안 주인공이 살아있도록 해야 마을 지키기에 성공한 것이다. 코인 기능이 있어서 'I'를 누르면 주인공의 목숨이 3개 증가한다.

프로젝트의 최상위 구조는 main이 있는 JFrame을 상속받은(extends) GateFrame 클래스이다. 모든 이미지는 JPanel을 사용하지 않고, GateFrame 위에 JLabel로 추가(add)하는 방식으로 설계했다. 그리고 게임에서 중요한 객체인 주인공, 몬스터, 차원문을 위한 Player, Enemy, Gate 클래스가 있다. Player와 Enemy의 벽 충돌 감지를 위해서 Runnable 인터페이스를 구현한 BackGroundPlayerService, BackGroundEnemyService 클래스를 만들었다. Movable 인터페이스는 Player와 Enemy의 움직임 상태를 위한 것이다. Player의 좌우 방향 확인을 위한 PlayerDirection 열거형 클래스도 있다. 부수적인 클래스로는 BGM는 게임 플레이 중 나오는 음악을 위한 클래스, Runnable 인터페이스를 구현한(텍스트 내용을 1초마다 바꾸기 위해) CountDown은 게임 플레이 시간을 알려주는 클래스, Head는 주인공 목숨을 위한 클래스, InsertCoin은 코인을 넣는 방법을 알려주는 텍스트를 띄우기 위한 클래스, Runnable 인터페이스를 구현한(목숨이 0이 되면 게임 종료 메시지를 띄우기 위해) LifeCount는 목숨의 개수를 위한 클래스, StartBGM은 시작했을 때 대기화면에서의 음악을 위한 클래스이다.

GateFrame은 main이 있는 클래스이다. 변수들부터 살펴보면 차원문 개수인 MAXGATES와 차원문이 생성되는 주기인 GATECOOLTIME을 정적 상수로 선언했다. backgroundMap은 배경화면 이미지를 위한 JLabel형 변수다. player, gate, enemy 이 3개는 객체 생성을 위한 변수이다. life, head, coin, sBGM도 객체(텍스트/이미지/음악) 생성을 위한 변수다. gateList와 enemyList는 gate와 enemy 객체들을 관리하기 위해서 Arraylist로 만들었다. enter는 키보드의 엔터 눌렀는지 확인하기 위한 변수다. main에서는 GateFrame 객체 하나를 생성하고, 생성자를 보면 initSetting(), initListener(), pressEnterToStart()를 호출한 다음 setVisible(true)를 통해 화면에 띄워준다. 먼저 initSetting()에서 배경화면 설정과 시작배경음악(sBGM)을 재생시킨다. initListener()에서는 이벤트리스너를 만든다. 키를 누를 때와 뗄 때의 이벤트 처리만 하면 되므로 KeyAdapter를 이용해서 키보드 클릭 이벤트 핸들러와 키보드 해제 이벤트 핸들러를 등록해준다. keyPressed() 핸들러에는 5가지 경우가 있다. 왼쪽, 오른쪽, 위, 스페이스 바, 엔터, I이다. 왼쪽(혹은 오른쪽) 키를 누르면, 주인공이 left(혹은 right)가 참이 아니면서 leftWallCrash(혹은 rightWallCrash)가 아닌 경우에만 left(혹은 right)가 실행한다. 위 키를 누르면, up이 거짓이면서 down도 거짓일 때 up을 실행한다. 스페이스 바를 누르면, 주인공의 attack()을 실행하고, gateList에 있는 마지막 항목의 gate의 BeingAttacked()를 실행한다. 엔터가 누르면, 화면과 음악을 변경(기존 음악 끄기)함으로써 게임을 시작한다. I를 누르면, 목숨을 3개 증가한다. 키보드 해제 이벤트

핸들러에서는 왼쪽, 오른쪽, 스페이스 바가 해제되는지 봐준다. 왼쪽(혹은 오른쪽) 키를 떼면 left(혹은 right)를 false로 바꾸고, leftStop(혹은 rightStop)을 호출한다.
pressEnterToStart()를 실행할 때는 시작 대기화면을 container에 덧붙이고, 스레드가 시작되는데, 엔터가 눌리는지 이벤트리스너가 감시해주고 있으므로, 엔터가 눌려서 enter가 참이 되면 객체들을 생성한다. initMapObject()에서 배경화면을 바꿔주고, initObject()를 통해서 private 변수들을 생성하고, 이들을 add()를 통해서 배경화면 위에 덧붙여준다. 그리고 게임 플레이 시간인 60초를 카운트 다운하는 스레드와 목숨 스레드를 시작한다.
GateFrame의 마지막 메서드인 initThread()는 gate와 enemy를 생성하고 스레드를 시작한다. MAXGATES 개수만큼만 gate를 만들고 GATECOOLTIME 초만큼 sleep을 거는 스레드를 시작한다. 게임 시작과 동시에 일단 gate 객체를 하나 만들고(player와 상호작용하기 위해서 생성자에 player 전달), 그 객체를 gateList에 추가한다. gateList[0]에 gate를 넣은 셈이다. 그리고 gate 스레드를 시작한다. 5초가 지난 뒤에는 gate 스레드가 시작되는 것까지는 동일한데, enemy를 생성할지 말지도 구현해야 한다. 따라서 gateList의 사이즈가 2일 때부터 확인하면 된다. 게임 시작한 지 10초가 지났다고 가정하면 gateIdx는 0이고, gateList의 해당 인덱스(0번) gate의 Hp가 0보다 클 때만 그 gate의 정보를 Enemy 생성자로 전달해서 enemy 객체를 생성한다. 이 객체도 enemyList에 추가하고, 화면에 추가하고, enemy 스레드까지 시작하면 된다. 게임 설정상 5초(GATECOOLTIME) * 12개(MAXGATES)는 60이므로, 마지막 차원문은 5초가 남았을 때 생성된다. 따라서 이 마지막 차원문을 못 부순 경우에도 enemy를 생성해주면 끝난다.
 메인을 제외하고, 주요 클래스로는 Player, Enemy, Gate가 있고, Runnable 인터페이스를 구현한 BackGroundPlaerService와 BackGroundEnemyService 클래스는 주인공과 몬스터의 벽 충돌 감지를 위해서 필요하다. Player 클래스는 Movable 인터페이스를 구현해줘야 한다. 우선 생성자에서는 initObject()를 통해서 주인공의 움직임 상태에 따른 이미지들을 설정해준다. initSetting()에서는 초기 조건들을 설정한다. 위치, 움직임 상태, 벽 충돌 상태, 공격 상태, 방향이 있다. initBackGroundPlayerService()에서는 스레드를 만들고 시작하는데, 메인스레드가 있는 GatreFrame은 키보드 이벤트 처리하느라 바쁘기 때문에 필요하다. 벽 충돌 감지 방법은 player나 enemy나 비슷하다. Runnable 인터페이스를 구현했기 때문에 run() 메서드를 오버라이드한다. BackGroundPlayerService 클래스에서 생성자에서는 색깔 판별을 위한 이미지를 하나 불러와 줘야(ImageIO.read()) 한다. 게임 플레이 중 보이는 허공은 흰색(R: 255, G: 255, B:255), 1층, 천장, 벽은 빨간색(R: 255, G: 0, B: 0), 1층을 제외한 땅은 파란색(R: 0, G: 0, B: 255)이다. run() 메서드에서는 주인공의 좌표를 기준으로 바닥에 충돌했는지, 벽에 충돌했는지 매우 빠르게(0.01초 sleep하는 스레드) 감시해주고, 그에 알맞은 작업들을 해준다(왼쪽/오른쪽/천장 벽에 충돌하면 각각에 맞는 WallCrash를 true로 바꾸고, 왼쪽, 오른쪽의 경우 벽을 뚫지 못하도록 그 방향으로의 주인공 움직임 상태를 false로 바꾼다). 주인공의 좌하단과 우하단의 좌표의 합을 적절히 계산해서 bottomColor를 정수형으로 두고 이게 -2이면 바닥을 의미한다. 따라서 바닥이 땅이면 down상태를 거짓으로 바꾸고, 땅이 아닌 허공이면 if문으로 점프하지 않았을 때만 down()이 실행되도록 만들어야 한다. 이 if문이 있어야 정상적인 점프가 가능하다. 다시 Player 클래스로 돌아와서 attack()은 스페이스 바를 눌렀을 때 실행되는 메서드이다. 공격 상태를 true로 바꾸고, 스레드를 람다식으로 구현(Runnable task 정의)하고 시작한다. 좌우에 따라서 이미지가 다르니까 if문으로 나눠야 하고, 0.17초 sleep을 걸어준다. Movable의

메서드들 중 먼저 left()를 구현했다. attack()도 비슷하게, left() 안에 단순히 움직임 상태를 참으로 바꾸고 이미지를 바꾸는 작업만 하는 것이 아니라, 스레드도 구현해야 한다. GateFrame에 메인 스레드만 있으면, 두 개의 키가 들어오면 이벤트가 하나씩 처리되므로 동시작업이 불가능하기 때문이다. 예를 들어서 오른쪽 키를 계속 누르고 있다가 위 키를 누르는 경우 메인 스레드만 있으면 오른쪽으로 이동하면서 계속 점프하는 게 아니라 오른쪽으로 이동하다가 한 번 점프하고 멈춰버린다. right()도 left()와 비슷하게 구현했다. up()과 down()은 좀 다르게 구현했다. up은 우선 움직임 상태(up)을 참으로 바꾸고, 스레드를 실행하는데, left()처럼 while이 아닌 for문으로 구현한다. 게임 설정상 중력이 있고, 이단점프 같은 건 불가능하기 때문이다. for문 조건식이 점프력이 되고, 그만큼 점프했으면 for문을 빠져나와서 up상태를 거짓으로 만들고 down()을 호출해야 한다. down()에서는 left()와 비슷하게 구현하는데, down() 키를 누를 일은 없으니까 while을 빠져나오면 down상태를 거짓으로 바꿔준다.

Enemy 클래스는 Player 클래스와 비슷하다. right()와 up()은 필요 없으므로 본체 구현을 하지 않았고, 왼쪽 벽 충돌과 땅 감지만 감지하면 되므로, BackGroundEnemyService에서 leftCrashWall만 확인해주었고, bottomColor를 통해서 땅 충돌을 감지해준다. 게임 설정상 enemy가 왼쪽 벽을 뚫고 나가면 몬스터의 이미지를 `setIcon(null)`해서 없앴고, 주인공의 목숨이 하나 감소하도록 하였다. 또한, 왼쪽 벽에 충돌했을 때 메모리 효율을 위해서 스레드를 종료하도록 구현했다. 차원문에서 나오는 몬스터를 랜덤함수를 이용해서 쫄병, 족장 중 랜덤하게 결정돼서 나오도록 구현했다.

마지막으로 Gate 클래스이다. 생성자에서 gate 객체를 인자로 받아야 한다는 점이 Enemy 클래스와 비슷하다. main에서 enemy 객체가 생성되는 초기 좌표를 설정하기 위함이었다. gate 객체를 만들 때는 player 객체를 인자로 받아야 하는데, 주인공이 공격 동작을 할 때 차원문이 주인공의 공격 범위에 있을 때만 유효하도록 구현해야 하기 때문이다. 생성자에서는 메인에서 전달한 player 정보를 받고, initObject()를 호출해서 hp가 5인 gate 이미지를 만들고, initSetting()을 통해서 디폴트 위치, 체력을 설정해준다. main에서 gate.start()를 호출하면 Gate 클래스의 run()이 실행된다. 차원문이 땅 범위 내의 랜덤한 좌표에서 생성되도록 Random함수를 사용했다. 랜덤하게 층(y좌표)이 결정되게 하고, 그에 맞춰서 if문을 통해 분기하고, 해당 층에 맞춰서 x좌표도 랜덤하게 결정되도록 했다. 차원문의 위치를 위에서 결정된 x, y 좌표를 통해 초기화해준다. 차원문이 생성되고 7초가 지나면 차원문 이미지를 삭제하고 메모리 효율을 위해 스레드를 멈췄다. 마지막 메서드는 BeingAttacked()이다. 스페이스 바를 누르면 이벤트 리스너는 이 메서드를 호출하게 되는데, 주인공과 차원문의 좌표 관계가 적절하고, 주인공의 공격 상태가 true일 때만 이 스레드가 실행되도록 while의 조건문에 추가해줬다. 이 과정에서 Math.abs()를 사용해서 좌표 차이의 절댓값을 확인했다. try catch문으로 0.17초 sleep을 걸었는데, 이 시간은 Player.attack()이 실행될 때 걸리는 sleep 시간과 같다. 다음으로 차원문의 체력이 0보다 큰지 아닌지 판단해서 0보다 크면 체력을 1 감소시킨다. 다음 if문에서는 먼저 체력이 0일 때인지 체크하는데, 만약 0이면 체력이 0일 때의 차원문 이미지로 교체하고 0.5초 뒤에 이미지를 없애고 메모리 효율을 위해 스레드를 멈춘다. 만약 체력이 0이 아닌 경우에는 각 체력에 맞는 차원문 이미지로 변경해주면 된다.

이 정도로 큰 규모의 프로젝트는 처음이라서 클래스들의 전체적인 구조를 생각하면서 코딩하는 게 쉽지 않았다. 프로젝트 진행 초반에는 스레드 관련 내용을 이해하는 데에 꽤

많은 시간이 들었던 것 같다. 그림 변경 시점은 이벤트 루프의 모든 작업이 완료되고 나서 repaint된다는 점, player 좌우 동작과 점프 동작에 있어서 메인 스레드만 있으면 두 개의 키가 들어올 때 이벤트가 하나씩 실행돼서 동시작업이 불가능하고, 따라서 Player.java에서 left() 등의 메서드 안에도 스레드가 필요하다는 점 등을 이해하는 것이 중요했던 것 같다. 객체서비스 클래스의 디테일을 구현하는 것도 수치가 조금만 벗어나도 버그가 생겨서 적절한 수치를 찾는 것도 중요했던 것 같다. 전담으로 도트를 찍어서 새로운 이미지를 만드는 팀원이 있었는데, 직접 해보지는 않았지만, 새로운 이미지 하나 만드는 것도 상당한 시간과 노력이 필요하다고 들었다. 게임 속 모든 이미지를 직접 만들기도 했고, 중간에 수정하거나 안 쓰는 이미지들도 꽤 많았기 때문에, 디자인적인 부분도 까다로웠다고 생각한다. 객체 간의 상호작용을 구현하는 것도 어려웠다. 모든 개념을 처음 배우면서 새로운 걸 구현해야 했기 때문에 이 부분에서 시간을 많이 썼다. 우선 생성자에서 다른 객체를 인자로 받아야 한다는 것부터 어디에서 각 객체의 메서드나 스레드를 어떻게 실행해야 할지까지 고민도 많았고, 시행착오도 많았다. 한 클래스의 객체를 여러 개를 만들려면 ArrayList를 사용한다는 것부터 시작했다. 그리고 생성된 객체를 화면에 덧붙여서 main에서 add()를 써서 이미지를 생성했다. 문제는 차원문(gate)과 적(enemy)을 어떻게 구현할까였다. 처음에는 일단 gate 객체를 만들고 스레드를 시작해봤다. 이 과정에서는 5초마다 차원문이 한 개씩만 보이는 걸 목표로 했다. 새로 생성된 차원문들은 gateList에서 관리되도록 했다. 따라서 `gateList.remove(0)`과 같이 5초가 지난 후에 ArrayList 요소 하나를 제거하는 작업도 필요했었다. 다음 작업으로 가장 큰 문제가 enemy 객체를 어떻게 생성하고 스레드를 실행할까였다. 오랜 고민과 구글링 끝에 Timer라는 개념을 써서 버그가 좀 많지만, 구현했었다. 버그를 잡아보려 하다가 점점 이상해져서 교수님께 조언을 구했다. 자세한 코드는 모르셨지만, 이때 교수님의 조언이 프로젝트 진행에 많은 도움이 되었다. Timer로 객체를 조종하지 말라고 하셨었다. 실제 시간과 맞지 않고, 코드가 꼬인다고 하셨었다. 메인에서 게임 시작과 동시에 모든 객체가 시작되는 방향으로 코드를 짜라고 조언해주셨었다. 조금 막막하긴 했지만, 일단 Timer는 쓰지 말아야겠다는 생각으로 다시 생각해봤다. 고민 끝에 내린 결론은 “enemy도 enemyList로 관리해서 그냥 5초 전에 생성된 차원문을 바라보자“였다. 코드도 훨씬 간단해질 것 같은 느낌으로 다시 코딩하였다. 처음에 gate만 생각하고 만들었던 코드에서 5초 전 gate를 if문을 통해 체력이 남았는지 봐주고, 남았다면 그냥 그 gate의 위치에서(그 gate를 enemy 객체 생성 시에 생성자 인자로 넘김으로써) enemy 객체를 생성하고 스레드를 실행시키기만 하면 끝이었다. 코드가 훨씬 깔끔해졌었다. `gateList.remove(0)` 같은 작업도 필요 없어졌었다. 키보드 클릭 이벤트 핸들러에서 스페이스 바가 눌리면, gateList의 요소 중 가장 마지막(size-1의 인덱스)의 gate의 체력이 0보다 큰 경우에만 BeingAttacked()를 호출하면 됐었다. Gate.BeingAttacked()에서도 player의 정보가 필요해서 Gate 생성자에서도 player를 인자로 받으면 됐었다. 이 외에도 자잘한 버그를 고치기 위해서 꼼꼼한 구현이 필요했던 것 같다.

아쉬운 점은 크게 두 가지 있는 것 같다. player와 enemy의 상호작용이 없다는 점과 stage가 하나뿐이라는 점이다. 막바지에 두 개 모두 구현해보려고 노력해봤었다. player와 enemy 간의 상호작용은 시간이 좀 더 있다면 가능할 것 같다. player와 gate 간의 상호작용인 BeingAttacked()와 비슷할 것 같다고 생각한다. stage 구현은 노력해봤는데, 감이 잘 안 잡혔었다. 교수님께서 Tile과 Floor를 이용하면 가능하다고 하셨었다. 나중에 기회가 되면 시도해보면 좋을 것 같다.