**컴퓨터공학설계및실험1 최종 프로젝트(Brick out) 보고서**

**20191244 김현승**

1. **프로젝트 목표**

이 프로젝트는 OpenFramework를 사용해 벽돌 깨기 게임을 구현해 봄으로써 OpenFramework 사용에 익숙해지고, 객체지향프로그래밍과 자료구조 및 알고리즘 숙련을 위해 계획되었다.

1. **실험 환경**

IDE : Visual Studio 2022 Debug x64

CPU : Intel i7-7700k

RAM : 16GB

OS : Window 10 Education

1. **소스 코드 파일 별 각 변수 및 함수에 대한 설명**

* **ball.h / ball.cpp**
* 상수 BALL\_RADIUS : 공의 반지름
* 상수 MAX\_BOUNCE\_ANGLE : 공이 패들에 닿았을 때 튕기는 최대 각도(라디안)
* float x, y : 공의 좌표
* float speedX, speedY : 공의 속도 X, Y 성분
* 메서드 readyUpdate : 게임 상태가 “ready”일 때 공이 패들 위에 붙어서 움직이도록 공의 위치를 업데이트한다.
* 메서드 update : 게임 상태가 “playing”일 때 공과 패들, 벽돌, 벽면 등과의 충돌을 감지하고 처리한다. 공의 위치와 속도 정보를 업데이트한다.
* 메서드 shoot : “ready” 상태에서 우클릭 시 공이 발사되도록 속도와 위치를 조절한다.
* 메서드 draw : 공의 위치를 반영해 그린다.
* **brick.h / brick.cpp**
* 상수 BRICK\_WIDTH : 벽돌의 너비
* 상수 BRICK\_HEIGHT : 벽돌의 높이
* 매크로 max, min : a와 b 중 큰/작은 값을 반환한다.
* float x, y : 벽돌의 왼쪽 모서리 좌표
* int life : 벽돌의 남은 수명. 공과 충돌할수록 감소한다.
* ofColor color : 벽돌의 색상. 벽돌의 색에 따라 수명이 달라진다.
* bool isAlive : 벽돌이 깨졌는지 남아있는지에 대한 정보
* 메서드 checkCollision : 공과 벽돌과의 충돌을 감지한다. 충돌 감지시 true 리턴.
* 메서드 collision : 공과 충돌했을 때 life와 isAlive, color를 갱신한다.
* 메서드 draw : 벽돌의 위치와 색을 반영해 벽돌을 그린다.
* **paddle.h / paddle.cpp**
* 상수 PADDLE\_WIDTH : 패들의 너비
* 상수 PADDLE\_HEIGHT : 패들의 높이
* 매크로 max, min : brick의 것과 동일하다.
* float x, y : 패들의 왼쪽 모서리 좌표
* 메서드 setX : 패들의 x좌표를 설정한다.
* 메서드 checkCollision : 공과 패들과의 충돌을 감지한다. 충돌 감지시 true 리턴
* 메서드 draw : 패들의 위치를 반영해 패들을 그린다. 패들의 y좌표는 항상 고정되어 있다.
* **rank.h / rank.cpp**
* Rank는 랭크 정보를 담고 있는 클래스이다.
* string name : 해당 랭크 정보의 등록자 이름
* int score : 해당 랭크 정보의 점수
* **stage.h / stage.cpp**
* Stage는 스테이지의 전반적인 정보를 모두 담고 있는 클래스이다
* int stageNumber : 해당 스테이지의 번호
* int aliveBrick : 해당 스테이지에 깨지지 않은 벽돌 개수
* vector<Brick> bricks : 해당 스테이지의 모든 벽돌 정보
* vector<Rank> ranks : 해당 스테이지에 기록된 모든 랭크 정보
* 메서드 addBrick : 스테이지에 벽돌 추가
* 메서드 addRank : 스테이지에 랭크 정보 추가
* 메서드 sortRanks : 퀵정렬을 이용해 랭크를 점수의 내림차순으로 정렬
* 함수 partition : 주어진 Rank 벡터를 가장 오른쪽 원소(high) 기준으로 왼쪽에는 큰 값, 오른 쪽에는 작은 값이 오도록 정렬
* 함수 quickSort : partition 함수를 이용해 피벗을 기준으로 Divide-and-Conquer 방식으로 전체 벡터를 정렬
* 메서드 readRanks : 텍스트 파일을 읽어 저장된 랭크 정보 불러오기
* 메서드 writeRanks : 갱신된 랭크 정보를 텍스트 파일로 출력
* **ofApp.h / ofApp.cpp (기본 메서드 제외)**
* bool isRestart : “start”로 돌아갈 때 재시작 여부 저장
* int currentStage : 현재 플레이 중인 스테이지 번호 -1
* int score : 현재 점수
* int nameCount : 랭크 기록을 위해 이름을 입력할 때 현재까지 입력한 글자 수
* rankPage : 랭크 열람 시 현재 페이지 정보
* string name : 랭크에 기록되는 입력된 이름
* vector<Stage> stages : json 파일로부터 로드된 스테이지 정보
* Paddle paddle : 패들 객체
* Ball ball : 공 객체
* string gameState : 현재 게임 상태
* float windowWidth, windowHeight : 게임 실행 화면의 크기
* 메서드 loadStages : json 파일을 읽어 stages 벡터에 정보를 저장하는 함수
* 메서드 loadOnlyBricks : json 파일에서 벽돌 정보만 읽어 각 stage에 저장하는 함수
* 메서드 drawUI : 게임 플레이시 우측에 위치하는 UI를 그리는 함수

1. **플로우차트, 자료구조 및 알고리즘, 시간/공간 복잡도**

* Flow Chart

텍스트, 도표, 평면도, 스크린샷이(가) 표시된 사진

자동 생성된 설명

위는 여러 클래스의 상호작용과, gameState에 따른 프로그램 흐름 변화를 나타내고 있다.

* 자료구조 및 알고리즘 / 시간, 공간 복잡도

이 프로그램은 전체적으로 5개의 클래스의 상호작용으로 실행되고 있다. 각 스테이지의 벽돌 위치 및 색상과 랭크 정보를 Stage 클래스로 관리하고, 플레이 도중 공과 패들 그리고 벽돌과의 충돌을 Paddle, Ball, Brick 클래스와의 상호작용으로 처리한다. 충돌이 발생했을 시 공의 적절한 속도 변화를 주어 반사를 구현했다. 또, 벽돌과 충돌 시는 해당 벽돌의 정보 변화가 일어나 벽돌이 깨지거나, 수명이 줄어들게 구현하였다. 이 프로그램에서 사용된 알려진 알고리즘은 “퀵정렬” 알고리즘이다. 매번 클리어 또는 게임오버시 랭크 정보를 작성하게 되는데, 이 때 작성된 랭크를 포함해 기존에 있던 랭크까지 점수의 내림차순으로 정렬한다. 다음은 Stage 클래스에서 퀵 정렬의 구현이다.

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

partition 함수에서 high에 해당하는 원소를 피벗으로 정해 피벗보다 큰 원소는 왼쪽, 작은 원소는 오른 쪽에 위치하도록 원소의 배열을 바꾼다. 피벗을 기준으로 원소가 양쪽으로 모두 나누어 졌다면, 피벗이 위치한 인덱스를 리턴하고, quickSort 함수에서 해당 피벗 인덱스를 기준으로 양쪽에 대해 다시 quickSort를 수행한다. 계속 재귀적으로 호출하다보면 low가 high보다 크거나 같아지는, 즉 원소가 1개 이하만 남게 될 때까지 진행된다. 여기서 알 수 있듯이 퀵 정렬은 큰 문제를 작게 나누어 해결하는 Divide-and-Conquer의 대표적인 알고리즘이다. 퀵 정렬 알고리즘은 정렬 알고리즘 중 실질적인 실행 속도가 가장 빠른 것으로 알려져있다. 평균적인 경우 O(nlogn)의 시간 복잡도를 갖게 되지만(partition에서 n번만큼의 반복과 logn번만큼 재귀호출), 피벗이 최댓값이나 최솟값으로 선택되는 경우 분할이 불균형하게 일어난다. 따라서 O(n^2)의 시간 복잡도를 가질 수 있다. 퀵 정렬은 재귀적으로 구현되기 때문에 이는 메모리의 스택 공간을 차지하게 된다. 평균적으로 분할이 잘 이루어진다고 생각하면, log n개만큼의 depth가 발생할 수 있으므로 O(logn)의 공간 복잡도를 갖지만, skewed tree 형태로 분할이 이루어진다면 O(n)의 공간 복잡도를 가질 수 있다.

1. **창의적인 구현**

내가 생각하는 창의적인 구현 부분은 공의 충돌과 속도 업데이트 부분의 로직 구현이다. 이는 Ball 클래스의 update 메서드와 Paddle, Brick 클래스의 checkCollision 메서드를 이용해 처리된다. 먼저 충돌을 판단하는 checkCollision 메서드이다.

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

Brick와 Paddle의 checkCollision 메서드는 첫 부분(isAlive 확인 분기)를 제외하곤 완전히 동일하다. 벽돌/패들의 가장자리와 공의 중심 사이 가장 가까운 점을 찾아 그 점과 공의 중심 사이 거리를 계산한다. 만약 이 거리가 공의 반지름보다 작거나 같게 된다면 공이 벽돌/패들의 가장자리와 접했거나 통과했다는 뜻이므로 충돌로 판단하게 된다. 여기서는 거리를 계산할 때 굳이 제곱근을 씌울 필요가 없기 때문에 거리의 제곱과 반지름의 제곱을 비교했다.

다음은 패들과 공이 충돌했을 때 공이 튀는 방향을 설정하는 로직이다. 벽돌 깨기 게임에서는 공이 벽돌이나 벽에 닿았을 때는 정확히 입사각과 반사각이 같도록 공이 반사되지만, 패들에 닿았을 때는 그렇지 않다. 패들의 중앙 부분과 가깝게 닿을수록 위쪽(-y) 방향으로 반사되고, 왼쪽이나 오른쪽 부분과 가깝게 닿을수록 해당 방향 쪽으로 반사된다. 다음은 Ball 클래스의 update 메서드의 일부이다.

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

패들과 충돌이 일어났다고 판단되었을 때, 패들의 중앙에서 공까지의 거리를 계산하고 이를 -1.0과 1.0 사이의 수로 바꾼다. 예컨대, 패들의 왼쪽에 가깝게 부딪혔을 경우 hitPoint는 -1.0에 가까워지고, 오른쪽에 가깝게 부딪혔을 경우 1.0에 가까워진다. 중앙에 가깝게 부딪힐수록 0에 가까운 값을 가질 것이다. 이 hitPoint를 최대 반사 각도인 MAX\_BOUNCE\_ANGLE과 곱해 반사각 bounceAngle을 계산한다. 패들의 끝 부분에 가깝게 튕길수록 더 반사각이 크게 만들어지는 것이다. 이를 이용해 speedX를 다시 계산하면 자연스러운 패들과의 반사 작용을 구현할 수 있다.

마지막으로, 벽돌과 충돌했을 때 반사각 계산 로직이다. 벽돌과의 충돌은 공이 벽돌의 측면에 부딪혔는지, 정면이나 후면에 부딪혔는지 확인해야한다. 다음은 Ball 클래스의 update 메서드의 일부이다.

텍스트, 스크린샷, 디스플레이, 폰트이(가) 표시된 사진

자동 생성된 설명

현재 스테이지의 모든 벽돌에 대해 충돌을 판단하고, 충돌된 벽돌이 감지되면 해당 벽돌과 공의 위치관계를 조사하게 된다. 먼저, 공이 벽돌의 측면에 모두 감싸지는지(?) 확인한다. 만약 그렇다면 공은 측면에 튕긴 것이므로 속도의 X성분만 반전시켜 수평 반사를 일으킨다. 그렇지 않다면 공이 벽돌의 전면 또는 후면에 모두 감싸지는지 확인한다. 그렇다면 공은 벽돌의 전면 혹은 후면에 튕긴 것이므로 속도의 Y성분만 반전시켜 수직 반사를 일으킨다. 속도 업데이트 이후 벽돌의 수명 life와 isAlive를 적절히 조정하고, 해당 스테이지에 남아있는 벽돌 수도 업데이트 해준다. 충돌이 감지된 벽돌에 대해 충돌 처리를 마친 이후에는 다른 벽돌에 중복적인 충돌이 발생할 수 있으므로 break로 루프를 종료하도록 했다.

위와 같은 패들, 벽돌과의 충돌 처리를 마친 뒤 다른 상단/측면 벽과의 충돌 처리도 해주고 현재 속도에 대해 공의 위치를 업데이트 해주면 한 번의 공 정보 업데이트가 완료되게 된다.

덧붙이자면, 스테이지 벽돌 정보를 json 파일로 관리하고 읽어들인 것도 꽤 좋은 방법이었다고 생각한다. “기초인공지능” 수업에서 진행한 과제 중 대부분에서 처리할 데이터 셋이 json 파일로 제공되었고, 이를 이용하는 것에 익숙해졌다. 이를 통해 이번 프로젝트에서 json 이용해 깔끔하고 체계적으로 스테이지 데이터셋을 관리할 수 있었다.

1. **프로젝트 실행 결과**

스크린샷, 디자인이(가) 표시된 사진

자동 생성된 설명

gameState가 “start”일 때 화면

스크린샷, 텍스트, 도표, 라인이(가) 표시된 사진

자동 생성된 설명

Start Game 버튼을 눌러 “ready” 상태의 화면

스크린샷, 도표, 텍스트, 직사각형이(가) 표시된 사진

자동 생성된 설명

스크린샷, 텍스트, 도표, 라인이(가) 표시된 사진

자동 생성된 설명

마우스 오른쪽 버튼을 눌러 공이 발사된 “playing” 상태의 화면

스크린샷이(가) 표시된 사진

자동 생성된 설명

공이 바닥으로 떨어져 “gameover” 상태가 된 화면. 이름 입력 후 랭크에 기록된다.

텍스트, 스크린샷, 도표, 폰트이(가) 표시된 사진

자동 생성된 설명

시작 화면에서 View Ranks 버튼을 눌러 “rank” 상태의 화면.

텍스트, 스크린샷, 도표, 라인이(가) 표시된 사진

자동 생성된 설명

A, D, Q 등의 입력으로 페이지(스테이지)를 넘기거나 메뉴로 돌아갈 수 있다.

1. **느낀 점 및 개선 사항**

이번 프로젝트를 진행하면서 객체지향프로그래밍의 재미와 편의성을 크게 느꼈다. 벽돌 깨기 게임에서 구현해야할 것들을 클래스로 구현하면서 객체간의 상호 작용으로 대부분을 처리하도록 구현했다. 모두 클래스로 나누어 구현하다보니 코드의 수정과 업데이트가 매우 편리했다. 수업 시간에 배운 “캡슐화”의 중요성이 크게 와 닿았던 부분이다.

더 구현하고 싶은 기능들이 많았지만 시간 상의 이유로 생략한 부분이 많다. 가장 아쉬운 것은 플레이어가 스테이지를 “커스텀”할 수 있는 기능을 구현하지 못 한 것이다. 마우스로 벽돌 위치를 지정해 배치하고 스테이지를 구성해 직접 플레이할 수 있는 기능을 추가하면 좋을 것 같다. 또, 난이도 기능을 추가해도 괜찮을 것 같다. 공의 전반적인 속도를 늘리거나(공 발사시 Y 성분 속도를 -5가 아닌 더 낮은 값으로 지정) MAX\_BOUNCE\_ANGLE을 늘림으로써 공이 더 크게 튕겨나가 제어가 힘들도록 할 수도 있다.