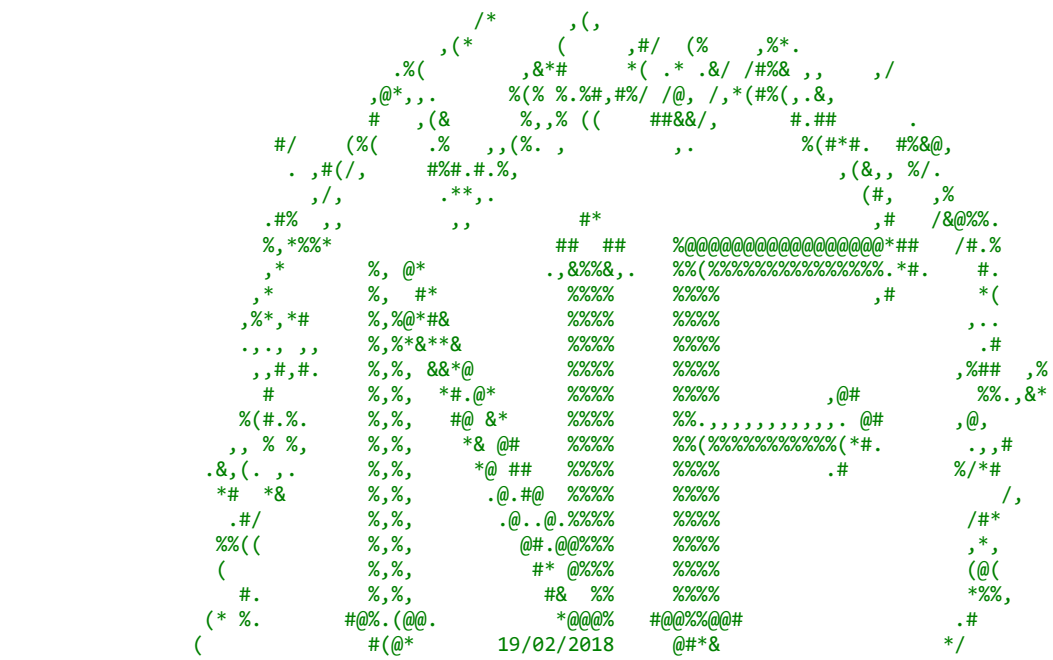

```
/*
#pragma once
#include "GamePlay.h"
#include "ControllableAvatar.h"
#include "CaptureArea.h"
#include "Projectile.h"


class Score
{
private:
	//Variables
	static const int numOfPlayers = 4;
	int groundScoreAccuracy;
	int xOnGrid;
	int yOnGrid;
	Colour* ground;
	int groundTeamScore[START_COLOUR + 1];
	int captObjTeamScore[numOfPlayers];
	int teamScores[numOfPlayers];
	float teamScoresPercentage[numOfPlayers];
	CaptureArea* captObjects;	//Here we will store all the capturable game objects
	int numOfCaptObjects = 0;
	static vector<Vector3> projPOS;
	static vector<float> projSize;
	static vector<Colour> projColour;


	//Functions
	void BuildScore();
	void BuildCaptObjectScore();
	void BuildGroundScore(); //Builds the array for the ground score



	void UpdateCapturableObjectScore(); //Updates the capturable objects
	void UpdateGroundScore(Avatar* player); //Updates the ground cells
```

```
        void UpdateScoreForProjectiles();

        // updates the ground score
        void ChangeGridScore(Colour teamToDecrease, Colour teamToIncrease);


public:
        Score();
        Score(static const int dimensionX, static const int dimensionY);
        Score(static const int dimensionX, static const int dimensionY, static const int
groundScoreAccuracy);
        void UpdateScores(); // Call To Update every available score

        static void UpdateProjectilesVector(Vector3 pos, Colour c, float size);

        // To Return Team Score
        inline int GetTeamScore(int id) { return teamScores[id]; }
        //To return Team Score Percentage
        inline float GetTeamScorePercentage(int id) { return teamScoresPercentage[id]; }

        void PrintScore(int x); // debug

        ~Score();

};


#include "Score.h"
#include "Game.h"

vector<Vector3> Score::projPOS;
vector<float> Score::projSize;
vector<Colour> Score::projColour;

Score::Score()
{
        groundScoreAccuracy = 10;
        xOnGrid = (int)2 * (20 + 10)*groundScoreAccuracy;
        yOnGrid = (int)2 * (20 + 10)*groundScoreAccuracy;
        BuildScore();
}

Score::Score(static const int dimensionX, static const int dimensionY) {
        groundScoreAccuracy = 100;
        xOnGrid = dimensionX;
        yOnGrid = dimensionY;
        BuildScore();
}

Score::Score(static const int dimensionX, static const int dimensionY, static const int
groundAccuracy) {
        groundScoreAccuracy = groundAccuracy;
        xOnGrid = dimensionX;
        yOnGrid = dimensionY;
        BuildScore();
}
```

```
//Build Score categories
void Score::BuildScore() {
	BuildGroundScore();
	BuildCaptObjectScore();
}

void Score::BuildGroundScore() {
	ground = new Colour[xOnGrid * yOnGrid *groundScoreAccuracy];
	for (int i = 0; i < xOnGrid * yOnGrid *groundScoreAccuracy - 1; i++) {
		ground[i] = START_COLOUR;
	}

	ground[0] = RED;
	ground[1] = GREEN;
	ground[2] = BLUE;
	ground[3] = PINK;
	for (int i = 0; i < 4; i++)
	{
		groundTeamScore[i] = 0;
	}
}

void Score::BuildCaptObjectScore() {
	vector<PhysicsNode*>* obj = PhysicsEngine::Instance()-
>GetAllNodesOfType(PAINTABLE_OBJECT);
	if (obj) {
		for (std::vector<PhysicsNode*>::iterator itr = obj->begin(); itr != obj-
>end(); ++itr) {
			captObjects = dynamic_cast<CaptureArea*>((*itr)->GetParent());
			numOfCaptObjects++;
		}

		delete obj;
	}
}

// Call To Update every available score
void Score::UpdateScores() {
	int maxScore = 0;

	for (int i = 0; i < numOfPlayers; i++) {
		if (Game::Instance()->GetPlayer(i)) {
			UpdateGroundScore(Game::Instance()->GetPlayer(i));
		}
	}
	if (numOfCaptObjects > 0) {
		UpdateCapturableObjectScore();
	}
	if (!projPOS.empty() && (!projSize.empty()) && (!projColour.empty())) {
		UpdateScoreForProjectiles();
	}
	for (int i = 0; i < numOfPlayers; i++)
	{
		teamScores[i] = (int)(groundTeamScore[i]) + captObjTeamScore[i];
		maxScore += teamScores[i];
	}

	for (int i = 0; i < numOfPlayers; i++) {
```

```
			teamScoresPercentage[i] = teamScoresPercentage[i] / maxScore;
		}
}

void Score::UpdateGroundScore(Avatar* player) {

		if (player->IsPlayerInAir()) {
			return;
		}

		float halfxGrid = xOnGrid / 2;
		float halfyGrid = yOnGrid / 2;

		Vector2 playerPos = Vector2((player->GetPosition().x *groundScoreAccuracy) +
(halfxGrid + (10 * groundScoreAccuracy)), (player->GetPosition().z *groundScoreAccuracy +
halfyGrid + (10 * groundScoreAccuracy)));

		float plGridSize = player->GetLife() * groundScoreAccuracy * 15;
		float radius = plGridSize * plGridSize;

		// Runs through the square arount the center and finds the circle.
		for (int i = playerPos.x - plGridSize; i <= playerPos.x; i++) {
			for (int j = playerPos.y - plGridSize; j <= playerPos.y; j++) {
		// check if inside vector boundries
				if ((i > 0 && j > 0) && (i + j < xOnGrid * yOnGrid - radius)) {
					if ((i - playerPos.x)*(i - playerPos.x) + (j - playerPos.y)*
(j - playerPos.y) <= radius) {
		// new position based on 2 dimensional position
						int posi = i * (xOnGrid);
						int posj = i * (xOnGrid)+j;
						int xSym = playerPos.x* (xOnGrid)-(posi - playerPos.x*
(xOnGrid));

						int ySym = i * (xOnGrid)+playerPos.y - (posj - (i *
(xOnGrid)+playerPos.y));
		// Thanks to symetry we take all 4 quadrants of the circle arount the center
						ChangeGridScore(ground[posi + posj], player-
>GetColour());
						ground[posi + posj] = player->GetColour();
						ChangeGridScore(ground[posi + ySym], player-
>GetColour());
						ground[posi + ySym] = player->GetColour();
						ChangeGridScore(ground[xSym + posj], player-
>GetColour());
						ground[xSym + posj] = player->GetColour();
						ChangeGridScore(ground[xSym + ySym], player-
>GetColour());
						ground[xSym + ySym] = player->GetColour();
					}
				}
			}
		}
}

// Decreases the score from previous team and increases the score to the new team.
void Score::ChangeGridScore(Colour teamToDecrease, Colour teamToIncrease) {
		groundTeamScore[teamToDecrease] -= 1;
		groundTeamScore[teamToIncrease] += 1;
```

```
}

// TODO: If needed update only the Capturable objects that got changed last frame
void Score::UpdateCapturableObjectScore() {
	for (int i = 0; i < numOfPlayers; i++)
	{
		captObjTeamScore[i] = 0;
	}

	for (int i = 0; i < numOfCaptObjects; i++) {
		switch (captObjects[i].GetColour())
		{
		case RED:
			captObjTeamScore[RED] += captObjects[i].GetScoreValue();
			break;
		case GREEN:
			captObjTeamScore[GREEN] += captObjects[i].GetScoreValue();
			break;
		case BLUE:
			captObjTeamScore[BLUE] += captObjects[i].GetScoreValue();
			break;
		case PINK:
			captObjTeamScore[PINK] += captObjects[i].GetScoreValue();
			break;
		default:
			break;
		}
	}
}

//It is called every update to update teh projectile score
void Score::UpdateScoreForProjectiles() {


	Vector3 pos = projPOS.back();
	projPOS.pop_back();
	float size = projSize.back();
	projSize.pop_back();
	Colour colour = projColour.back();
	projColour.pop_back();

	float halfxGrid = xOnGrid / 2;
	float halfyGrid = yOnGrid / 2;

	Vector2 pPos = Vector2((pos.x * groundScoreAccuracy) + halfxGrid, (pos.z *
groundScoreAccuracy) + halfyGrid);

	int plGridSize = size * groundScoreAccuracy * 10;
	int radius = plGridSize * plGridSize;

	// Runs through the square arount the center and finds the circle.
	for (int i = pPos.x - plGridSize; i <= pPos.x; i++) {
		for (int j = pPos.y - plGridSize; j <= pPos.y; j++) {
	// check if inside vector boundries
			if ((i > 0 && j > 0) && (i + j < xOnGrid * yOnGrid - radius)) {
				if ((i - pPos.x)*(i - pPos.x) + (j - pPos.y)* (j - pPos.y) <=
radius) {
```

```
	// new position based on 2 dimensional position
                                    int posi = i * (xOnGrid);
                                    int posj = i * (xOnGrid)+j;
                                    int xSym = pPos.x* (xOnGrid)-(posi - pPos.x*
(xOnGrid));
                                    int ySym = i * (xOnGrid)+pPos.y - (posj - (i *
(xOnGrid)+pPos.y));
	// Thanks to symetry we take all 4 quadrants of the circle arount the center
                                    ChangeGridScore(ground[posi + posj], colour);
                                    ground[posi + posj] = colour;
                                    ChangeGridScore(ground[posi + ySym], colour);
                                    ground[posi + ySym] = colour;
                                    ChangeGridScore(ground[xSym + posj], colour);
                                    ground[xSym + posj] = colour;
                                    ChangeGridScore(ground[xSym + ySym], colour);
                                    ground[xSym + ySym] = colour;
                                }
                            }
                        }
	}
}

//Creates a vector of all the Projectiles
void Score::UpdateProjectilesVector(Vector3 pos, Colour c, float size)
{
	projPOS.push_back(pos);
	projSize.push_back(size);
	projColour.push_back(c);
}

void Score::PrintScore(int score) {
	NCLDebug::Log(to_string(score));
}

Score::~Score()
{
}
```