Министерство образования Республики Беларусь

Учреждение образования

«Беларусский государственный университет информатики и радиоэлектроники»

Факультет компьютерных систем и сетей

Кафедра электронных вычислительных машин

Дисциплина: Системное программное обеспечение вычислительных машин

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовому проекту

на тему

FTP-клиент с возможностью получения/отправки архивированных каталогов

БГУИР КП 1-40 02 01 10 ПЗ

Студент: гр.050541 Кохановский Г.Е.

Руководитель: Поденок Л.П.

Минск 2023

# Введение

Язык С++ предназначен для разработки высокопроизводительного программного обеспечения. Язык является улучшенной и расширенной версией языка С, предназначенной для поддержки объектно-ориентированного программирования.

Цель курсового проекта исследовать возможности языка С++ для разработки и создания программного модуля, эмулирующего физику реального мира. Для решения задачи была выбрана двухмерная система координат, объекты, используемые для эмуляции – ПООП и круг. В симуляторе реализовано решение следующих задач в модели физической системы:

* Построение многообразия ПООП.
* Построение многообразия Круга.
* Разрешение коллизии.
* Стратегия импульсного разрешения.
* Использование нормали коллизии и глубины проникновения для определения импульса.
* Закон реституции Ньютона.
* Расчёт инвертной массы.
* Разрешение для проблемы погружения с поправкой на Floating point error IEEE754.

Физический движок — это симулятор твердого тела и его поведения в реальном времени. Некоторые физические движки довольно универсальны и полезны для обычных твердых тел и частиц; другие включают в себя различные соединители и ограничения, позволяющие моделировать ragdoll. Иные сосредоточены на мягких телах и жидкостях.

# Перечень используемых сокращений

1. ПООП ( AABB – axis-aligned bounding box) – параллельный осям ограничивающий параллелепипед. Это параллелепипед со сторонами, параллельными осям координат, ограничивающий некоторый геометрический объект в пространстве. При вращении объекта параллелепипед сохраняет свою ориентацию, однако может менять свои размеры.
2. СтСв (Degree of Freedom, DOF) – степень свободы. При постановке кинематической задачи необходимо распознать тип относительного движения, разрешенного в каждом из суставов, и присвоить ему некоторый переменный параметр (параметры) для измерения или расчета движения. Одной из первых проблем при проектировании или анализе механизма является количество степеней свободы, также называемое подвижностью устройства.

# Обзор Методов и Алгоритмов решения поставленной задачи

Реализуемый в рамках курсового проекта программный модуль базируется на реализации двумерной декартовой системы координат. Пространство физического движка представляет собой плоскость, заполняемую фигурами с конечной или бесконечной массой. В процессе эмуляции движения фигур происходит расчет взаимодействий фигур и реакции на столкновения. [7]

В работе эмулятора реализованы структуры, описывающие следующие типы фигур: Круг и ПООП.

При взаимодействии фигур первично происходит тест определения коллизии. Проверка двух кругов на возникновение коллизии происходит через сложение их радиусов и расчет расстояний между центральными точками окружностей. [1]

Для преодоления ошибки с проницаемостью фигур и восстановлением правильного скольжения используется метод расчета позиционной коррекции учитывающий порог, при достижении которого происходит расчет. [5]

Для визуализации программы используется набор библиотек GLUT. GLUT — это OpenGL Utility Toolkit, независимый от оконной системы набор инструментов для написания программ OpenGL. Он реализует простой оконный интерфейс программирования приложений (API) для OpenGL. [3]

# Обоснование выбранных методов и алгоритмов

Для построения виртуальной физической модели в курсовом проекте использованы и реализованы методы векторной математики для эмуляции взаимодействующих объектов и актов их взаимодействия.

В процессе взаимодействия двух объектов возникает коллизия , которая представляет из себя событие взаимодействия, пересечения двух объектов в системе координат. Для исследования (*раздел 9 п.3*) и определения (*раздел 9 п.1, п.2*) коллизии в курсовом проекте реализован подход, основанный на использовании параллельного осям ограничивающего параллелепипеда (далее ПООП). ПООП используется для расчета и обнаружения пересечения фигур, включенных в ПООП.

ПООП сложной формы можно использовать в качестве простого теста, чтобы увидеть, могут ли пересекаться более сложные формы внутри ПООП.

Структурное представление ПООП в проекте:

struct AABB

{

Vec2 min;

Vec2 max;

};

Эта форма позволяет представить ПООП двумя точками. Точка min представляет нижние границы осей x и y, а max представляет верхние границы, иначе говоря - значения представляют верхний левый и нижний правый углы.

Другим объектом взаимодействия является круг. Следующая структура описывает круг через значение радиуса и центральной точки:

struct Circle

{

float radius

Vec position

};

Проверить пересечение двух окружностей в системе координат возможно сложив их радиусы и сравнив полученное значение с расстоянием между центрами окружностей расположенных в системе координат (*раздел 9 п.2*).

Для разрешения коллизии используем стратегию импульсного разрешения.

Разрешение коллизии — это действие, при котором два объекта, которые оказались пересекающимися, модифицируются таким образом, чтобы они не оставались пересекающимися.

Импульсное разрешение — это особый тип стратегии разрешения коллизий.

Разрешая обнаруженные коллизии, мы накладываем ограничение на движение, так что объекты не могут оставаться пересекающимися друг с другом. Идея импульсного разрешения заключается в использовании импульса (мгновенного изменения скорости) для разделения обнаруженных сталкивающихся объектов. Для этого необходимо делать расчет с учетом массы, положения и скорости каждого объекта: большие объекты, сталкиваясь с меньшими, будут незначительно менять положение в пространстве, а маленькие объекты отлетать, значительно меняя положение в пространстве. Так же определим, что объекты с бесконечной массой будут неподвижны.

Степени Свободы определяют количество направлений, в которых объект ( ПООП или Круг) может перемещаться в системе координат. В двухмерном пространстве физического движка объект может двигаться с тремя степенями свободы: движение по XY-системе координат, что соответствует двум степеням, и так же вращение, что соответствует третьей степени свободы. В данном курсовом проекте для упрощения задачи исключается третья степень свободы – вращение, и рассматриваются только 2 степени свободы – движение в XY-системе координат.

Объекты заключенные в ПООП будут представлять твердое тело. Под твердым телом можно понимать объект образованный из набора из N частиц, если расстояние между любыми двумя частицами фиксировано и постоянно. ПООП и круги в рамках проекта будут рассматриваться как твердые тела, подверженные взаимодействию, и не подверженные деформации, то есть измнеение расстояний между N количеством частиц внутри твердого тела не происходит и не рассчитывается.

Для разрешения задачи о коллизии обратимся к двум параметрам:

* Нормаль коллизии.
* Глубина проникновения.

Нормаль коллизии — это направление, в котором будет применяться импульс.

Глубина проникновения определяет количественную величину импульса при воздействии одного объекта на другой.

Используя эти параметры при расчете коллизии объектов найдем вектор и силу импульса, с которой коллизия будет разрешена.

Для разрешения коллизии определим уравнение нахождения векторной величины импульса:

Составим уравнение длины вектора из позиции А до позиции В:

[ VAB = VB - VA ] 3.1

По сути, использовав разность endpoint - startpoint

Для определения относительной скорости движения между точками А и В добавим к уравнению нормаль коллизии ***n*** – это позволит вычислить относительную скорость движения от A до B вдоль направления нормали коллизии:

[ VAB ∙ n = (VB - VA ) ∙ n] 3.2

Используем скалярное произведение, как сумму покомпонентных произведений:   
[ , , ] 3.3

Следующим шагом разрешения коллизии является введение коэффициента реституции. Реституция характеризует эластичность или упругость объекта. Каждый объект в системе координат физического движка будет обладать реституцией, представленной числом в десятичной системе счисления. Поскольку при коллизии объектов каждый из них имеет значение реституции, для решения задачи необходимо выбрать одно из двух значений. Правильным выбором будет наименьшее из двух доступных значений реституции.

e = min( A.restitution, B.restitution )

Как только ***e*** определено, помещаем его в уравнение для определения величины импульса.

Обратимся к закону реституции Ньютона:

[V' = e × V ] 3.4

Скорость после столкновения равна скорости до него, умноженной на некоторую константу. Эта константа представляет собой «*bounce factor*». Зная это, становится довольно просто интегрировать реституцию в наш текущий вывод:

[ VAB ∙ n = -e × (VB - VA) ∙ n ] 3.5

В этом уравнении был введен отрицательный знак, указывающий направление. В законе реституции Ньютона (*V'*), результирующий вектор после отскока движется в направлении, противоположном V.

Теперь выразим скорости, находясь под воздействием импульса. Вот простое уравнение для изменения вектора некоторым импульсным скаляром (***j***) в определенном направлении относительно нормали (***n***):

[ V' = V + j \* n ] 3.6

Имея единичный вектор (***n***), который представляет собой направление, скаляр (***j***), представляющий длину вектора (***n***), затем добавляем масштабированный вектор (***n***) к (*V*), чтобы получить (*V'*). Это позволяет нам применить импульс одного вектора к другому.

Формально импульс определяется через изменение момента импульса. Момент импульса равен произведению массы и скорости. Зная это, представим импульс в том виде, в каком он формально определен, следующим образом:

[ *Impulse* = *mass* × *Velocity* => *Velocity* = => V' = V + ] 3.7

Далее выразим импульс с помощью (***j***) в отношении двух взаимодействующих объектов. При столкновении объектов A и B объект A будет отброшен в направлении, противоположном B:

[ V'A = VA + , V'B = VB - ] 3.8

Эти два уравнения будут отталкивать A от B вдоль единичного вектора направления (***n***) на величину скалярного импульса (***j***).

Объединяя уравнения 8 и 5 получим:

[ (VA - VV + + ) × n = -e × (VB - VA) ∙ n 3.9  
**=>** (VA - VV + + ) × n + e × (VB - VA) ∙ n = 0 ] 3.10

Имея задачу изолировать нашу величину, зная направление разрешения коллизии (задается событием обнаружения коллизии), остается определить величину этого направления. Величина, которая в нашем случае неизвестна, это (***j***) => мы должны изолировать (***j***) и найти её:

[ (VA - VV + + ) × n + e × (VB - VA) ∙ n = 0 3.12  
**=>** (1 + e)(( VB - VA) ∙ n) + j×( + ) × n = 0 3.13  
 **=>** j = ] 3.14

Так же для упрощения работы симуляции, в системе координат, если объекты удаляются друг от друга, разрешение коллизий не рассматривается.

Следующее что необходимо рассмотреть - инвертная масса вычисляется несколько раз, потому эффективным решением будет хранить инвертную массу для каждого объекта и предварительно вычислять ее один раз:

A.inv\_mass = 1 / A.mass

Нужно отметить, импульсный скаляр (***j***) распределяется по двум объектам. Таким образом, маленькие объекты отталкиваясь от больших объектов с большим значением импульса (***j***), а большие объекты изменяют свои скорости на незначительную величину импульса (***j***).

При текущей реализации симулятора объекты с нормальной массой при возникновении коллизии будут отскакивать друг от друга. Необходимо обработать сценарий, при котором объект может иметь бесконечную массу. Для симуляции выберем в качестве значения бесконечной массы – 0 (ноль). Такое решение приводит к необходимости переработать подход к получению инвертной массы, чтобы избежать деления на 0. Используем следующий подход:

if(A.mass == 0){

A.inv\_mass = 0}

else {

A.inv\_mass = 1 / A.mass }

Нулевое значение позволит выполнить решение правильно во время разрешения импульса.

Проблема погружения объектов возникает, когда объект А начинает погружаться в объект В под действием силы тяжести. Например, объект с низкой реституцией ударяется о стену с бесконечной массой и начинает погружаться в него.

Такое поведение является результатом ошибки при вычислениях с плавающей запятой. Во время каждого вычисления с плавающей запятой из-за несовершенства аппаратных средств возникает небольшая ошибка с плавающей запятой (Floating point error IEEE754). Со временем эта следствие ошибки накапливается в ошибке позиционирования, в результате чего объекты тонут друг в друге.

Для преодоления влияния ошибки используем метод, называемый позиционной коррекцией. Позиционная коррекция уменьшает проникновение двух объектов на небольшой процент, и выполняется после приложения импульса. Позиционная коррекция очень проста: переместим каждый объект по нормали коллизии (***n***) на процент от глубины проникновения.

В примененном подходе масштабируется значение PenetreDepth по общей массе системы. Это даст позиционную поправку, пропорциональную массе, с которой производится вычисление. Маленькие предметы отталкиваются быстрее, чем более тяжелые.

Дополнительным улучшением, исключающем возможное дрожание объектов, связанное с исправлением ошибки позиционирования является введение порога “slop” для применение коррекции при утопанные объектов.

Последняя рассматриваемая часть реализации системы - генерация простого многообразия. Многообразие в математических терминах можно описать как множество точек, представляющих область в пространстве, либо пространство, локально сходное с евклидовым. В нашем случае под многообразием будет пониматься объект, который содержит информацию о столкновении двух объектов.

Многообразие в системе описано структурой:

struct Manifold{

Object \*A;

Object \*B;

float penetration;

Vec2 normal;

};

В процессе симуляции при обнаружениях столкновений должны быть рассчитаны как проникновение, так и нормаль столкновения. Чтобы найти эти данные, исходные алгоритмы обнаружения столкновений должны быть расширены обрабатывая конкретные случаи взаимодействия объектов симуляции.

Рассмотрим простой алгоритм коллизии: Круг против Круга.

Разрешение коллизии будет происходить вдоль вектора от окружности А к окружности В. Глубина проникновения связана с радиусами кругов и расстоянием друг от друга. Перекрытие кругов можно вычислить, вычитая суммарные радиусы на расстояние от каждого объекта.

Разрешение коллизии ПООП против ПООП сложнее, чем Круг против Круга . Нормаль коллизии не будет вектором из A в B, а будет нормалью лицевой поверхности. ПООП представляет собой коробку с четырьмя гранями. Каждое лицевая поверхность имеет нормаль. Эта нормаль представляет собой единичный вектор, перпендикулярный грани.

Общее уравнение для линии в двумерном пространстве:

[ *ax + by + c* = 0 , *normal* = ]

В этом уравнении *a* и *b* являются нормальным вектором для линии, а вектор (***a, b***) предполагается нормированным (длина вектора равна нулю). Нормаль коллизии (направление разрешения столкновения) будет сонаправлена с одной из нормалей лицевой поверхности. В уравнении ***c*** это расстояние от начала координат. Это полезно использовать для проверки, находится ли точка на той или иной стороне линии.

Остается выяснить какая грань сталкивается на одном из объектов с другим объектом – так мы получаем нормаль. Однако иногда несколько граней двух ПООП могут пересекаться, например, когда два угла пересекаются друг с другом. Это означает, что мы должны найти ось наименьшего проникновения.

Последним случаем для рассмотрения является Круг против ПООП. Идея здесь состоит в том, чтобы вычислить ближайшую точку на ПООП к окружности; при таком подходе обработка случая сводится к схожему методу с «Круг против круга». Как только ближайшая точка вычислена и обнаружена коллизия, нормаль становится направлением ближайшей точки к центру круга. Глубина проникновения в данном случае — это разница между расстоянием от ближайшей точки к окружности и радиусом окружности.

Есть один сложный частный случай: если центр круга находится в пределах ПООП, то центр круга должен быть обрезан до ближайшего края ПООП, а нормаль необходимо отразить.

# Описание Алгоритмов Решения Задачи

1. Готовый тест для определения коллизии разработанный Кристофером Эриксоном:

Шаг 1. В функцию передаются ширина и высота в виде двух аргументов.

Шаг 2. Программа входит в конструкцию if.

Шаг 3. Сравниваем верхнее значение вектора, а по оси х с нижним значением вектора b по оси х.

Шаг 4. Сравниваем нижнее значение вектора, а по оси х с верхним значением вектора b по оси х.

Шаг 5. Производим дизъюнкцию результатов сравнения.

Шаг 6. В случае false переходим к Шаг 14.

Шаг 7. В случае true переходим к Шаг 8.  
Шаг 8. Программа входит в конструкцию if.

Шаг 9. Сравниваем верхнее значение вектора, а по оси y с нижним значением вектора b по оси y.

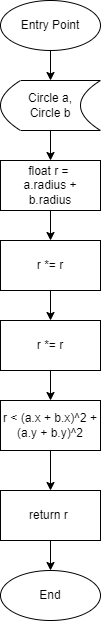
Шаг 10. Сравниваем нижнее значение вектора, а по оси y с верхним значением вектора b по оси y.

Шаг 11. Производим дизъюнкцию результатов сравнения.

Шаг 12. В случае false переходим к Шаг 14.

Шаг 13. Вернуть true.

Шаг 14. Выход из функции.

1. Проверка двух кругов на пересечение через их радиусы и центральные точки.  
   В качестве варианта улучшения, исключающего использование встроенной функции извлечения корня:
2. Простой метод разрешения коллизии:

Шаг 1. В функцию передаются объекты А и В.

Шаг 2. Вычисляем разность между скоростью А и В.

Шаг 3. Присваиваем результат вычисления переменной rv.

Шаг 4. Выполняем Скалярное произведение rv, normal.

Шаг 5. Присваиваем результат вычисления velAlongNormal.

Шаг 6. Программа входит в конструкцию if.

Шаг 7. Производим сравнение velAlongNormal > 0.  
Шаг 8. Для результата true переходим к Шаг 24.

Шаг 9. Используя функцию min получаем меньшее из значений A.restitution и B.restitution.

Шаг 10. Присваиваем наименьшее значение переменной е.

Шаг 11. Вычисляем выражение -(1 + e) \* velAlongNormal.

Шаг 12. Присваиваем значение переменной j.

Шаг 13. Вычисляем выражение 1 / A.mass + 1 / B.mass.

Шаг 14. Делим j на результат предыдущего вычисления.

Шаг 15. Присваиваем значение переменной j.

Шаг 16. Вычисляем j \* normal.

Шаг 17. Присваиваем результат вычисления impulse.

Шаг 18. Вычисляем 1 / A.mass \* impulse.

Шаг 19. Вычитаем из A.velocity результат вычисления.

Шаг 20. Присваиваем результат вычитания A.velocity.

Шаг 21. Вычисляем 1 / B.mass \* impulse.

Шаг 22. Суммируем с B.velocity результат вычисления.

Шаг 23. Присваиваем результат вычитания B.velocity.

Шаг 24. Выход из функции.

1. Diagram

   Description automatically generatedАлгоритм для разрешения коллизии Круг против Круга:
2. Полный алгоритм для генерации многообразия ПООП в ПООП и обнаружения столкновений:

Шаг 1. Вход в функцию AABBvsCircle.

Шаг 2. Передача аргумента Manifold \*m.

Шаг 3. Объявление и инициализация Object \*A = m->A.

Шаг 4. Объявление и инициализация Object \*B = m->B.

Шаг 5. Вычисление, объявление и инициализация Vec2 n = B->pos - A->pos.  
Шаг 6. Объявление и инициализация Vec2 closest = n.  
Шаг 8. Вычисление, объявление и инициализация float x\_extent = (A->aabb.max.x - A->aabb.min.x) / 2.

Шаг 9. Вычисление, объявление и инициализация float y\_extent = (A->aabb.max.y - A->aabb.min.y) / 2.

Шаг 10. Присвоение closest.x = Clamp( -x\_extent, x\_extent, closest.x ).

Шаг 11. Присвоение closest.y = Clamp( -y\_extent, y\_extent, closest.y ).

Шаг 12. bool inside = false.

Шаг 13. Если true - вычисление, объявление и инициализация float a\_extent = (abox.max.y - abox.min.y) / 2. Если false перейти к Шаг 32.

Шаг 14. Вычисление, объявление и инициализация float b\_extent = (bbox.max.y - bbox.min.y) / 2.

Шаг 15. Вычисление, объявление и инициализация float y\_overlap = a\_extent + b\_extent - abs( n.y ).

Шаг 16. Вход в if.

Шаг 17. Проверка выражения y\_overlap > 0.

Шаг 18. Если true - Вход в if. Если false перейти к Шаг 26.

Шаг 19. Проверка выражения x\_overlap > y\_overlap.

Шаг 20. Если true - Вход в if. Если false перейти к Шаг 23

Шаг 21. Проверка выражения n.x < 0.

Шаг 22. Присваивание m->normal = Vec2( -1, 0 ) и переход к Шаг 24

Шаг 23. Присваивание m->normal = Vec2( 0, 0 ).

Шаг 24. Присваивание m->penetration = x\_overlap.

Шаг 25. Возврат true и переход к Шаг 32.

Шаг 26. Вход в if.

Шаг 27. Проверка выражения n.y < 0.

Шаг 28. Если true присваивание m->normal = Vec2( 0, -1 ).

Шаг 29. Иначе присваивание m->normal = Vec2( 0, 1 ).

Шаг 30. Присваивание m->penetration = y\_overlap.

Шаг 31. Возврат true.  
Шаг 32. Выход из функции.

1. Обработка сценария если центр круга находится в пределах ПООП:

Шаг 1. Вход в функцию AABBvsAABB.

Шаг 2. Передача аргумента Manifold \*m.

Шаг 3. Объявление и инициализация Object \*A = m->A.

Шаг 4. Объявление и инициализация Object \*B = m->B.

Шаг 5. Вычисление, объявление и инициализация Vec2 n = B->pos - A->pos.

Шаг 6. Объявление и инициализация AABB abox = A->aabb.

Шаг 7. Объявление и инициализация AABB bbox = B->aabb.  
Шаг 8. Вычисление, объявление и инициализация float a\_extent = (abox.max.x - abox.min.x) / 2.

Шаг 9. Вычисление, объявление и инициализация float b\_extent = (bbox.max.x - bbox.min.x) / 2.

Шаг 10. Вычисление, объявление и инициализация float x\_overlap = a\_extent + b\_extent - abs( n.x ).

Шаг 11. Объявление и инициализация bool inside = false.

Шаг 12. Вход в if.

Шаг 13. Проверка выражения n == closest.

Шаг 14. Если true – присвоение inside = true. Иначе переход к Шаг 25.

Шаг 15. Вход в if.

Шаг 16. Вычисление и проверка выражения abs( n.x ) > abs( n.y ).

Шаг 17. Если true - Вход в if. Иначе переход к Шаг 21.

Шаг 18. Проверка выражения closest.x > 0.

Шаг 19. Если true - присвоение closest.x = x\_extent. Иначе переход к Шаг 20.  
Шаг 20. Присвоение closest.x = -x\_extent. Переход к Шаг 25.

Шаг 21. Вход в if.

Шаг 22. Проверка выражения closest.y > 0.

Шаг 23. Если true - присвоение closest.y = y\_extent. Иначе переход к Шаг 24.  
Шаг 24. Присвоение closest.y = -y\_extent.

Шаг 25. Инициализация, вычисление и присвоение Vec2 normal = n - closest.

Шаг 26. Инициализация, вычисление и присвоение real d = normal.LengthSquared( ).

Шаг 27. Инициализация, вычисление и присвоение real r = B->radius.

Шаг 28. Вход в if.

Шаг 29. Проверка выражения d > r \* r && !inside.

Шаг 30. Если true – return false. Переход к Шаг 39.

Шаг 31. Вычисление и присвоение d = sqrt( d ).

Шаг 32. Вход в if.  
Шаг 33. Проверка истинности inside.

Шаг 34. Если true - присвоение m->normal = -n. Иначе переход к Шаг 36.  
Шаг 35. Вычисление и присвоение m->penetration = r – d. Переход к Шаг 38.

Шаг 36. Присвоение m->normal = n.   
Шаг 37. Вычисление и присвоение m->penetration = r - d.  
Шаг 38. Return true  
Шаг 39. Выход из функции

1. Базовый метод расчета позиционной коррекции:

Шаг 1. Вход в функцию PositionalCorrection.

Шаг 2. Передача аргументов Object A, Object B.

Шаг 3. Объявление и инициализация Object \*A = m->A.

Шаг 4. Объявление и инициализация Object \*B = m->B.

Шаг 5. Объявление константы и инициализация const float percent = 0.2.

Шаг 6. Объявление, вычисление и инициализация Vec2 correction = penetrationDepth / (A.inv\_mass + B.inv\_mass)) \* percent \* n.

Шаг 7. Вычисление и присвоение A.position -= A.inv\_mass \* correction.  
Шаг 8. Вычисление и присвоение B.position += B.inv\_mass \* correction.

Шаг 9. Выход из функции.

1. И улучшенная версия, учитывающая порог при достижении которого происходит расчет:

Шаг 1. Вход в функцию PositionalCorrection.

Шаг 2. Передача аргументов Object A, Object B.

Шаг 3. Объявление и инициализация Object \*A = m->A.

Шаг 4. Объявление и инициализация Object \*B = m->B.

Шаг 5. Объявление константы и инициализация const float percent = 0.2.  
Шаг 6. Объявление константы и инициализация const float slop = 0.01

Шаг 6. Объявление, вычисление и инициализация Vec2 correction = max( penetration - k\_slop, 0.0f ) / (A.inv\_mass + B.inv\_mass))\*percent\*n.

Шаг 7. Вычисление и присвоение A.position -= A.inv\_mass \* correction.  
Шаг 8. Вычисление и присвоение B.position += B.inv\_mass \* correction

Шаг 9. Выход из функции.

# Заключение

Решая основную задачу курсового проекта - разработка и создание программного модуля, эмулирующего физику реального мира, я изучил научные и технические основания и инструменты для разработки такого рода продуктов. В процессе создания эмуляции научился работать с двухмерной системой координат, а также получил опыт работы с объектами в ней. Основной физической задачей взаимодействия объектов стало разрешение коллизии, для которой использовалась стратегия импульсного разрешения.

В заключении стоит заметить, что процесс симуляции физического мира, даже ограниченный простыми взаимодействиями, требует погружения в векторную математику, физику, математическую топологию, ставит задачу разработчику эффективно использовать возможности языка С++ для переноса методов решения в программный код.

# СПИСОК ЛИТЕРАТУРЫ

1. Gino Van Der Berger, Efficient Collision Detection of Complex Deformable Models using AABB Trees / Gino Van Der Berger - The Netherlands: November 6, 1998,
2. J. J. Uicker, Theory of Machines and Mechanisms / J. J. Uicker, G. R. Pennock, and J. E. Shigley - New York: Oxford University Press, 2003.
3. Mark J. Kilgard, The OpenGL Utility Toolkit (GLUT) Programming Interface // Mark J. Kilgard, Silicon Graphics, Inc., 1996
4. Ming-Hwa Wang, Collision Detection and Resolution / Ming-Hwa Wang - Santa Clara, CA: Department of Computer Engineering.
5. Randy Gaul, How to Create a Custom 2D Physics Engine: The Basics and Impulse Resolution // Randy Gaul Apr 6, 2013, URL: <https://gamedevelopment.tutsplus.com/>
6. Rigid Body Dynamics / MIT: OpenCourseWare, 2014. URL: <https://ocw.mit.edu>
7. David M.Bourg, Physics for Game Developers, Second Edition // David M. Bourg and Bryan Bywalec, CA: O’Reilly Media, Inc.
8. Герберт Шилдт, C++ Руководство для начинающих. Второе издание. / Герберт Шилдт, С.Н. Тригуб, Н.М. Ручко – М.: Издательский Дом Вильямс. 2005.