Министерство образования Республики Беларусь

Учреждение образования

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерных систем и сетей

Кафедра информатики

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовому проекту

на тему

«Игра в стиле Autochess»

Студент гр. 953502 Лавренович А. Е.

Руководитель ассистент кафедры информатики И.А.Удовин

Минск 2020

**Содержание**

[**Введение**](#ВВЕДЕНИЕ)

1. [**Анализ предметной области**](#АПО)
   1. [**Игра в стиле Autochess «Divine Rebellion»**](#З11)
   2. [**Выбор языка и средств разработки**](#З12)
   3. [**Постановка задачи**](#З13)
2. [**Разработка программы**](#РазрПрог)
   1. [**Реализация классов из предметной области**](#З21)
      1. [**Класс Unit**](#З211)
      2. [**Класс Melee**](#З212)
      3. [**Класс Ranged**](#З213)
      4. [**Класс Tile**](#З214)
      5. [**Класс BattleField**](#З215)
      6. [**Класс Missile**](#З216)
   2. [**Реализация основных алгоритмов программы**](#З22)
      1. [**Алгоритм поиска пути для воинов дальней дистанции**](#З221)
      2. [**Алгоритм поиска пути для воинов ближней дистанции**](#З222)
      3. [**Алгоритм выстрела и перезарядки (класс Ranged)**](#З223)
   3. [**Описание компонентов формы и обработчиков событий её компонентов**](#З23)
   4. [**Краткое описание текущей версии программы**](#З24)

[**Заключение**](#ЗАКЛЮЧЕНИЕ)

[**Список использованных источников**](#СПИСОКИСТ)

**ВВЕДЕНИЕ**

Первые видеоигры использовали множество электронных устройств для связи с пользователем. Так, первой зарегистрированной игрой, которую можно отнести к видеоигре, является Ракетный симулятор, интерфейсом которого выступало аналоговое устройство на базе электронно-лучевой трубки (ЭЛТ) (заявка на патент была подана 25 января 1947 года). Создание этой игры было обусловлено изобретением ЭЛТ, что позволяло формировать изображение на экране. Однако, из-за того, что Ракетный симулятор не имел растрового типа изображения, которое характерно для мониторов и телевизоров, то данное устройство чаще принимают за электронную игру, но относят к видеоиграм. Но вместе с тем, с технической точки зрения, Ракетный симулятор является видеоигрой.

Ранними примерами видеоигр являются первая программная игра OXO, первая созданная для развлечения *Tennis for Two, Spacewar!* как первая видеоигра, которая была доступна вне разработавшей её организации. Определение видеоигры для игр этого времени имеет свои трудности, например, данные игры не создавались как интерактивные программы, и не разрабатывались как игры. При этом они не были популярными и массовыми, так как не тиражировались и не выходили за пределы тех организаций, где они были созданы. Это обуславливалось рядом сложностей, когда компьютеры 1960-х занимали комнаты, использовали радиолампы, для которых требовалось охлаждение и постоянное обслуживание. Одна из революций в компьютерных играх заключалась в переходе на транзисторную элементную базу и массовое использование телетайпа. Здесь одним из следствий явилось разграничение игр по интерфейсу взаимодействия с пользователем: видеоигры и текстовые игры.

Исторически видеоигры появились раньше, чем был создан персональный компьютер. Это обусловлено тем, что в 1960-х годах компьютерные технологии не были массовыми и не рассматривались как серьезный инструмент в гражданской сфере. Создаваемые видеоигры на электронной базе (электронные игры) сыграли важную роль в мирной социализации компьютерной техники и её интеграции в популярную культуру. В 1970-х видеоигры стали одним из самых популярных развлечений, когда устанавливались игровые автоматы в барах, кафе и других заведениях. [[2]](#И2)

1. **АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ**
   1. **Игра в стиле Autochess «Divine Rebellion»**

***Dota Auto chess*** – стратегическая игровая модификация для компьютерной игры Dota 2. Разработана Drodo Studio и выпущена в январе 2019 года, модификация поддерживает до восьми игроков. Её популярность с более чем 8 миллионами игроков к маю 2019 года привела к признанию жанра «авто-баттлер», что привело к выпуску ряда других игр. Позже, в 2019 году, Drodo Studio разработала автономную версию, известную просто как Auto Chess, а Valve Corporation, разработчик Dota 2, разработала собственную автономную версию, известную как Dota Underlords.

Когда разработчиков спросили о вдохновении Dota Auto Chess, вскоре после выхода модификации, Drodo Studio заявила, что они черпают вдохновение из китайской игры Маджонг на основе тайлов для справки. Обычно в Маджонге играют четыре человека. Цель игры в Маджонг состоит в том, чтобы прописать определённую комбинацию карт с помощью ряда правил, не позволяя противникам создавать свои собственные комбинации. Игра фокусируется на технике, стратегии и расчетах, а также опирается на удачу. Черпая вдохновение из Маджонга, Dota Auto Chess внесла изменения, чтобы приспособить ориентированный на сражения геймплей.

С момента своего выпуска Drodo Studio обновила игру, добавив в неё настройки игрового баланса, исправления ошибок и обновления нового контента. Обновления контента включают в себя добавление и удаление багов, а также добавление новых рас в игру. С ростом популярности этой модификации, Valve Corporation, разработчик Dota 2, отправила Drodo Studio в свою штаб-квартиру, чтобы обсудить набор их для разработки автономной версии Dota Auto Chess. Valve и Drodo Studio пришли к выводу, что они не могут работать вместе напрямую, хотя они согласились, что будут создавать отдельные автономные адаптации игры и будут поддерживать друг друга. 1 июня 2019 года корпорация Valve обновила ежемесячный платный «Auto Chess Pass» в магазине Dota 2, а часть дохода пошла на Drodo Studio.[[2]](#И2)

На текущий момент существует довольно большое количество разнообразных клонов этой игры, и один из таких – Teamfight Tactics. Суть игр в жанре Autochess состоит в том, чтобы среди нескольких игроков за столом набрать наиболее сильную комбинацию из различных существ и не дать другим игрокам собрать комбинации, которые очень хорошо себя показывают против вашей. С каждым выигранным раундом вы уменьшаете здоровье противника и получаете деньги на покупку новых существ в магазине, однако существ там ограниченное количество и некоторые из них очень редки, а значит одновременно собрать похожую комбинацию двум игрокам сразу будет крайне проблематично. Пока ваше здоровье не равно нулю, вы можете участвовать в сражениях и драться за победу, однако как только всё здоровье заканчивается, то вы выбываете из матча, давая дорогу другим игрокам. Разумеется, победит выживший.

Нынешняя наша версия игры в таком жанре, **«Divine Rebellion»**, составляет сильно урезанную реализацию боевой системы, сердца самой игры, где находится только 2 типа существ, однако приложение довольно легко поддаётся дальнейшей разработке в силу своей архитектуры.

* 1. **Выбор языка и средств разработки**

Для написания курсовой работы выбран язык программирования **С#.**

**С#** (произносится си шарп) — объектно-ориентированный язык программирования. Разработан в 1998—2001 годах группой инженеров компании Microsoft под руководством Андерса Хейлсберга и Скотта Вильтаумота как язык разработки приложений для платформы Microsoft .NET Framework. Впоследствии был стандартизирован как ECMA-334 и ISO/IEC 23270. **С#** относится к семье языков с C-подобным синтаксисом, из них его синтаксис наиболее близок к **C++** и **Java**. Язык имеет статическую типизацию, поддерживает полиморфизм, перегрузку операторов (в том числе операторов явного и неявного приведения типа), делегаты, атрибуты, события, свойства, обобщённые типы и методы, итераторы, анонимные функции с поддержкой замыканий, LINQ, исключения, комментарии в формате XML.

Наиболее приятными особенностями языка являются делегаты, события, лямбда-выражения и интерфейсы (пришли на смену множественному наследованию из языка С++). Чётко разграниченная модель типов значений и ссылочных типов позволяет очень легко производить упаковку и распаковку объектов, рефлексия помогает с быстрым определением класса того или иного объекта с помощью метаданных, а особая реализация немедленной очистки памяти с помощью интерфейса IDisposable [[1]](#И1) позволяет наиболее эффективно расходовать память без резких скачков её заполнения (для графических приложений, использующих интревальную перерисовку, это критически важно).

Для реализации проекта использован инструмент **Windows Forms**. **Windows Forms** — интерфейс программирования приложений (API), отвечающий за графический интерфейс пользователя и являющийся частью Microsoft .NET Framework. Данный интерфейс упрощает доступ к элементам интерфейса Microsoft Windows за счет создания обёртки для существующего Win32 API в управляемом коде. Причём управляемый код — классы, реализующие API для Windows Forms, не зависят от языка разработки. То есть программист одинаково может использовать Windows Forms как при написании ПО на **C#**, **С++,** так и на **VB.Net**, **J#** и др.

С одной стороны, Windows Forms рассматривается как замена более старой и сложной библиотеке MFC, изначально написанной на языке C++. С другой стороны, WF не предлагает парадигму, сравнимую с MVC. Для исправления этой ситуации и реализации данной функциональности в WF существуют сторонние библиотеки. Одной из наиболее используемых подобных библиотек является User Interface Process Application Block, выпущенная специальной группой Microsoft, занимающейся примерами и рекомендациями, для бесплатного скачивания. Эта библиотека также содержит исходный код и обучающие примеры для ускорения обучения.

Внутри .NET Framework, Windows Forms реализуется в рамках пространства имён *System.Windows.Forms*. Было выбрано реализовывать проект на языке **C#** из-за его наиболее проработанной совместимости с .NET.

* 1. **Постановка задачи**

В задачу проекта входит разработать боевую систему видеоигры, используя C# и Windows Forms API. Для реализации понадобится:

* Реализовать иерархию классов воинов с одним общим абстрактным наследником для удобства хранения на поле;
* Реализовать специфические элементы каждого из действительных классов воинов;
* Создать класс поля битвы;
* Отобразить поле с воинами на графических компонентах формы;
* Снарядить пользователя возможностями выбора воинов, их выставления на поле и удаления с него, удобством использования графического интерфейса.

1. **Разработка программы**
   1. **Реализация классов из предметной области**
      1. **Класс Unit**

Абстрактный класс воина, где описываются необходимые поля для каждого возможного вида воинов на поле.

public enum Direction{ Left, Up, Right, Down }

public enum DamageType{ Magic, Physical }

public enum Team { Red, Blue }

public abstract class Unit: IDisposable

{

//IDisposable realization

public Bitmap Texture { get; protected set; }

public bool IsAlive { get; set; }

public bool HasMoved { get; protected set; }

public Direction Dir { get; protected set; }

public DamageType DmgType { get; protected set; }

public Team UTeam { get; protected set; }

public Unit Target { get; set; }

public int Health { get; protected set; }

public int MaxHealth { get; protected set; }

public int PhysDef { get; protected set; }

public int MagDef { get; protected set; }

public int AADmg { get; protected set; }

public bool IsAttacking { get; protected set; }

protected int UX, UY;//координаты

public Unit(Team team, int x, int y)

{

//constructor code

}

public abstract void Move(Tile[,] bt);

public void UnsetMove()

{

HasMoved = false;

}

public abstract bool SomeoneInRange(Tile[,] bt, int h, int w);

public abstract void AttackTarget(BattleField bf);

public void TakeDamage(BattleField bf, int dmg, DamageType dmt)

{

//selected target takes damage

}

protected void SetTarget(Unit t)//найти подходящую цель

{

Target = t;

}

}

* + 1. **Класс Melee**

Класс воина ближнего боя, который имеет возможность атаковать на одну клетку в 4-х направлениях: вверх, вниз, вправо, влево. Двигается такой воин на одну клетку за ход и из возможностей имеет только автоатаки.

public class Melee: Unit, ICloneable

{

//IDisposable & ICloneable realization

private Stack<int> px, py; //стеки наиболее благоприятных ходов

private int[,] wayMap; //карта для алгоритма Ли

public Melee(Team team, int x, int y) : base(team, x, y)

{

//constructor code

}

public override bool SomeoneInRange(Tile[,] bt, int h, int w)

{

//определить, находится ли кто-то в зоне поражения автоатакой

}

public override void Move(Tile[,] bt)

{

//передвинуться на клетку с наименьшим весом (движение к цели)

}

public override void AttackTarget(BattleField bf)

{

//Цель получает урон, если ещё не мертва

}

private void InititalizeWayMap(Tile[,] bt)

{

//инициализирует карту для работы алгоритма поиска пути на каждом юните

}

private void CreateNewWayMap()//lee algo

{

//данный алгоритм создаёт новую карту волновым алгоритмом Ли и обновляет //стеки с наиболее благоприятными координатами последовательного //перемещения к цели, по которым в итоге двигается воин

}

}

* + 1. **Класс Ranged**

Класс воина дальнего боя, который имеет возможность атаковать на любую дистанцию во всех 4-х направлениях. Имеет особенность перезарядки, т.е. стреляет не каждый ход, а через некоторое их количество. По мере определения цели в радиусе поражения начинает атаковать, если заряжен. Атака происходит следующим образом: в направлении, в котором находится цель, вылетает снаряд, являющийся объектом класса Missile, после чего воин начинает перезаряжаться. Может ходить во время перезарядки. Использует упрощённую версию алгоритма поиска пути перемещения. Ходит на одну клетку за ход.

public class Ranged: Unit, ICloneable

{

//IDisposable & ICloneable realization

private int movesOnReload;

private int moveCount;

public Ranged(Team team, int x, int y): base(team, x, y)

{

//constructor code

}

public override bool SomeoneInRange(Tile[,] bt, int h, int w)

{

//определить, находится ли кто-то в зоне поражения

}

private bool InPossibleRange(Tile[,] bt, int h, int w, int rx, int ry)

{

//вспомогательная функция для нахождения пути

}

public override void Move(Tile[,] bt)

{

//Ищет наиболее благоприятный вариант клетки, чтобы переместиться

//через нахождение наименьшей дистанции до любого противника, однако

//просматривает, будет ли возможность атаковать при перемещении на одну из

//четырёх клеток, используя InPossibleRange. Если будет такая возможность,

//то воин без дальнейших раздумий переместится туда

}

public override void AttackTarget(BattleField bf)

{

//атакует врага, выпуская в его направлении снаряд

}

private double Distance(int x1, int y1, int x2, int y2)

{

//обычный расчёт дистанции между двумя точками

}

}

* + 1. **Класс Tile**

Класс клетки на боевом поле (класс BattleField). Предназначен для хранения информации о снарядах и воинах в соответсвующих полях. На одной клетке может находиться несколько снарядов, однако только один воин. Для хранения множества снарядов используется коллекция List<T> [[1]](#И1).

public class Tile: ICloneable

{

public Unit Warrior { get; set; }//воин, занимающий клетку

public bool IsFree { get; set; }//показывает, пуста ли клетка

public List<Missile> Missiles;

public Tile()//инициализация новой пустой клетки

{

Warrior = null;

IsFree = true;

Missiles = new List<Missile>();

}

public object Clone()//необходимо для клонирования состояния поля перед битвой

{

return new Tile

{

Warrior = (Unit)((this.Warrior == null) ? null : ((this.Warrior is Melee) ? ((Melee)this.Warrior).Clone() : ((Ranged)this.Warrior).Clone())),

IsFree = this.IsFree,

Missiles = new List<Missile>()

};

}

}

* + 1. **Класс BattleField**

Класс, предназначенный для управления всеми объектами на игровом поле Tile[,] BattleTiles, а также реализующий событие смерти воина, чтобы убрать его с поля. Реализует метод начала битвы между 2-мя командами, блокируя пользовательский интерфейс на время битвы. Когда битва кончается, на экран выводится сообщение о команде победителя, либо же о ничье. Для поддержки битвы используется компонент *System.Windows.Forms.Timer*, где и задаётся интервал между ходами. [[4]](#И4)[[5]](#И5)

public class TileEventArgs : EventArgs//нужен для события OnDied

{

public int X { get; set; }

public int Y { get; set; }

}

public class BattleField

{

System.Windows.Forms.Timer BattleTimer;

private PictureBox FieldBox;//то, что выводится на экран через привязку к форме

private Bitmap CleanField;//текстура абсолютно чистого поля

private Bitmap InitField;//текстура поля, которая отображает боевые действия

public static readonly int resolution = 75;

public string UnitChoice { get; set; }

public Tile[,] BattleTiles { get; private set; }

private Tile[,] BTCopy { get; set; }

private bool allowMouseMove;//флаг разрешения события MouseMove на поле

private bool allowMouseClick;//флаг разрешения события MouseClick на поле

public static readonly int width = 900, height = 750;

readonly Pen Pen = new Pen(Color.Yellow, 5);//ей рисуется квадратик выбора клетки

readonly Pen HPen = new Pen(Color.Red, 3);//кисть, которой рисуется HealthBar

public EventHandler<TileEventArgs> OnDied;//событие для обработки умершего воина

private string toShow;

public BattleField()

{

//constructor code

}

private void FreeBCell(object sender, TileEventArgs e)

{

//обработчик события смерти на клетке воина

}

private void FieldBox\_MouseMove(object sender, MouseEventArgs e)

{

//обработчик события MouseMove на поле

}

private void FieldBox\_MouseClick(object sender, MouseEventArgs e)

{

//обработчик события MouseClick на поле

}

public static Bitmap ResizeImage(Image image, int new\_height, int new\_width)

{

//ресайз изображения, чтобы его размер совпадал с клиентской областью

}

public void LoadTexture(Bitmap bmp, int x1, int y1, Direction dir, bool doesAttack)

{

//загрузка текстуры на поле

}

public void ClearCell(int x1, int y1)//очищает клетку от изображения

{

//очищает клетку от изображения

}

public void ClearFullField(bool cleanObjects)//стирает всё, оставляя фон поля

{

//стирает всё, оставляя фон поля (если cleanObjects == true, то также удаляет все объекты с поля)

}

private void Redraw()

{

//перерисовка поля после очередного хода

}

private void TimerEventProcessor(object obj, EventArgs e)

{

//обработчик тика таймера, реализует ход на поле

}

private bool CheckFightDone()

{

//проверяет, не закончилась ли битва, каждый ход

}

public void StartFight()

{

//метод, начинающий битву на этом поле

}

}

* + 1. **Класс Missile**

Класс снаряда, выпускаемого воином дальней дистанции. Снаряд выпускается в направлении выбранного на момент атаки противника и больше никогда не меняет своей траектории. Снаряд может вылететь за пределы поля. Снаряд содержит атрибут команды, которой он принадлежит, из-за чего не может ранить своих, однако он тут же нанесёт урон и исчезнет при встрече хоть какого-то противника. На одной клетке может быть одновременно несколько снарядов.

public class Missile

{

public Bitmap Texture { get; private set; }

public Direction MDir { get; private set; }

public DamageType DmgType { get; private set; }

public Team MTeam { get; private set; }

private int Damage { get; set; }

private int MX { get; set; }

private int MY { get; set; }

public bool HasMoved { get; private set; }

private Unit Target { get; set; }

public Missile(Team team, Direction dir, int x, int y, DamageType dmgt, int dmg)

{

//constructor code

}

public void Act(BattleField bf, Tile[,] bt)

{

//определяет действие, производимое на этом ходе снарядом (Attack или Move)

}

public void Attack(BattleField bf)

{

//если цель не мертва, атакует её, после чего исчезает

}

public void Move(Tile[,] bt, int h, int w)

{

//перемещение на одну клетку вперёд по направлению

}

private bool SomeoneInRange(Tile[,] bt, int h, int w)

{

//проверка на возможность поражения цели (в районе одной клетки по напр.)

}

private void SetTarget(Unit t)

{

Target = t; //установка цели для атаки

}

public void UnsetMove()

{

HasMoved = false; //обновление флага хода

}

}

* 1. **Реализация основных алгоритмов программы**
     1. **Алгоритм поиска пути для воинов дальней дистанции**

***Краткое описание реализации алгоритма поиска и построения кратчайшего пути для воинов дальней дистанции***

Для реализации используется алгоритм вычисления дистанции между двух точек. Сам алгоритм поиска находит наименьшую дистанцию до любого противника при возможном перемещении воина на клетку, т.е. если после перемещения туда дистанция до какого-то врага минимальна, то она станет приоритетной клеткой, куда и пойдёт воин. Однако, больший приоритет имеет клетка, при перемещении на которую воин тут же может атаковать.

private bool InPossibleRange(Tile[,] bt, int h, int w, int rx, int ry)

{

for (int y = ry - 1; y >= 0; y--) // SEARCH UPWARDS

if (!bt[y, rx].IsFree && bt[y, rx].Warrior != null && bt[y, rx].Warrior.UTeam != this.UTeam)

return true;

//блок кода выше повторяется для каждого из 4-х направлений

return false;

}

public override void Move(Tile[,] bt)

{

if (HasMoved)

return;

if ((moveCount + 1) % movesOnReload != 0)

moveCount++;

IsAttacking = false;

List<int> listCoordsX = new List<int>();//х-координаты каждого противника

List<int> ListCoordsY = new List<int>();//у-координаты каждого противника

int rx = UX, ry = UY;

int h = BattleField.height / BattleField.resolution, w = BattleField.width / BattleField.resolution;

for (int y = 0; y < h; y++)

for (int x = 0; x < w; x++)

if (!bt[y, x].IsFree && bt[y, x].Warrior != null && bt[y, x].Warrior.UTeam != this.UTeam)

{

listCoordsX.Add(x);

ListCoordsY.Add(y);

}

double mindist = 1000;//минимальная дистанция до цели

//для более удобной работы уничтожаем списки, переводя их в вид массивов

int[] coordsX = listCoordsX.ToArray(), coordsY = ListCoordsY.ToArray();

listCoordsX.Clear();

ListCoordsY.Clear();

//сам алгоритм поиска клетки с наименьшим весом, сначала посмотрим ситуацию //на месте, не двигаясь

int n = coordsX.Length;

double tmp;

int bestx = UX, besty = UY;

for (int i = 0; i < n; i++)

{

tmp = Distance(rx, ry, coordsX[i], coordsY[i]);

if (tmp < mindist)

mindist = tmp;

}//ON PLACE

//затем посмотрим ситуацию на каждой из 4-х возможных для перемещения клетках

ry = UY - 1; //UPWARDS

rx = UX;

if (ry >= 0 && ry < h && bt[ry, rx].IsFree)

{

if (InPossibleRange(bt, BattleField.height / BattleField.resolution, BattleField.width / BattleField.resolution, rx, ry))

{

bt[ry, rx].Warrior = this;

bt[ry, rx].IsFree = false;

bt[UY, UX].Warrior = null;

bt[UY, UX].IsFree = true;

UY = ry;

UX = rx;

HasMoved = true;

Dir = Direction.Up;

return;

}

for (int i = 0; i < n; i++)

{

tmp = Distance(rx, ry, coordsX[i], coordsY[i]);

if (tmp < mindist)

{

mindist = tmp;

bestx = rx;

besty = ry;

Dir = Direction.Up;

}

}

}

//блок кода выше повторяется для каждого из 4-х направлений

if (bestx != UX || besty != UY)//перемещаемся

{

bt[besty, bestx].Warrior = this;

bt[besty, bestx].IsFree = false;

bt[UY, UX].Warrior = null;

bt[UY, UX].IsFree = true;

UY = besty;

UX = bestx;

}

HasMoved = true;

}

private double Distance(int x1, int y1, int x2, int y2)

{

return Math.Sqrt(Math.Pow(x1 - x2, 2) + Math.Pow(y1 - y2, 2));

}

* + 1. **Алгоритм поиска пути для воинов ближней дистанции**

***Краткое описание реализации алгоритма поиска и построения кратчайшего пути для воинов ближней дистанции***

Для реализации используется одна из разновидностей *алгоритма Ли*, где наиболее благоприятные клетки записываются в два поля-стека Stack<int> px, py. Отсюда позднее будут браться необходимые координаты. На вершине стека будут находиться те самые координаты, в которые следует переместиться воину на очередном ходу. Ниже сама реализация: [[6]](#И6)

private void InititalizeWayMap(Tile[,] bt)//инициализирует карту для алгоритма

{

for (int i = 0; i < BattleField.height / BattleField.resolution; i++)

for (int j = 0; j < BattleField.width / BattleField.resolution; j++)

if (bt[i, j].IsFree)

wayMap[i, j] = -1;

else

{

if (bt[i, j].Warrior == null)

wayMap[i, j] = -2;//wall

else if (bt[i, j].Warrior != null && bt[i, j].Warrior.UTeam != this.UTeam)

wayMap[i, j] = -3;//possible target

else if (bt[i, j].Warrior != null && bt[i, j].Warrior.UTeam == this.UTeam)

wayMap[i, j] = -2;//act like it's a wall

}

}

private void CreateNewWayMap()//сам алгоритм

{

int h = BattleField.height / BattleField.resolution;

int w = BattleField.width / BattleField.resolution;

wayMap[UY, UX] = 0;//start will be zero

px.Clear();

py.Clear();

int[] dx = { 1, 0, -1, 0 }; // смещения, соответствующие соседям ячейки

int[] dy = { 0, 1, 0, -1 }; // справа, снизу, слева и сверху

int d, x, y, k;

bool stop;

d = 0;

do

{

stop = true; // предполагаем, что все свободные клетки уже помечены

for (y = 0; y < h; y++)

for (x = 0; x < w; x++)

if (wayMap[y, x] == d)// ячейка (x, y) помечена числом d

{

for (k = 0; k < 4; k++)//проходим по непомеченным соседям

{

int iy = y + dy[k], ix = x + dx[k];

if (iy >= 0 && iy < h && ix >= 0 && ix < w && wayMap[iy, ix] == -1)

{

stop = false; // найдены непомеченные клетки

wayMap[iy, ix] = d + 1; // распространяем волну

}

}

}

d++;

} while (!stop);

// восстановление пути

int len = 1000; // длина кратчайшего пути из (ax, ay) в (bx, by)

int bx = 0;

int by = 0;

for (y = 0; y < h; y++)

for (x = 0; x < w; x++)

if (wayMap[y, x] == -3)// ячейка (x, y) помечена числом d

{

for (k = 0; k < 4; k++)// проходим по всем непомеченным соседям

{

int iy = y + dy[k], ix = x + dx[k];

if (iy >= 0 && iy < h && ix >= 0 && ix < w && wayMap[iy, ix] < len && wayMap[iy, ix] >= 0)

{

bx = ix;

by = iy;

len = wayMap[iy, ix];

}

}

}

d = len;

while (d > 0)

{

px.Push(bx);

py.Push(by);// записываем ячейку (x, y) в путь

d--;

for (k = 0; k < 4; ++k)

{

int iy = by + dy[k], ix = bx + dx[k];

if (iy >= 0 && iy < h && ix >= 0 && ix < w && wayMap[iy, ix] == d)

{

bx = bx + dx[k];

by = by + dy[k];//переходим в ячейку, которая на 1 ближе к старту

break;

}

}

}

}

* + 1. **Алгоритм выстрела и перезарядки (класс Ranged)**

***Краткое описание реализации алгоритма выстрела***

Реализация состоит в следующем: выпустить снаряд, создавая экземпляр объекта Missile с нужными характеристиками и помещая его на поле боя, где он будет двигаться, пока не попадёт в цель или не вылетит с поля. Однако если на текущий ход воин ещё не зарядился, то он не может стрелять, и просто ждёт, обновляя счётчик. Ниже представлена его реализация:

public override void AttackTarget(BattleField bf)

{

//выпустить снаряд с нужными характеристиками в нужном направлении

if (Target != null)

{

if ((moveCount + 1) % movesOnReload != 0)

{

moveCount++;//это поле отвечает за перезарядку

return;

}

moveCount = 0;

switch (Dir)

{

case Direction.Down:

bf.BattleTiles[UY + 1, UX].Missiles.Add(new Missile(this.UTeam, Dir, UX, UY + 1, DmgType, AADmg));

break;

case Direction.Up:

bf.BattleTiles[UY - 1, UX].Missiles.Add(new Missile(this.UTeam, Dir, UX, UY - 1, DmgType, AADmg));

break;

case Direction.Left:

bf.BattleTiles[UY, UX - 1].Missiles.Add(new Missile(this.UTeam, Dir, UX - 1, UY, DmgType, AADmg));

break;

default:

bf.BattleTiles[UY, UX + 1].Missiles.Add(new Missile(this.UTeam, Dir, UX + 1, UY, DmgType, AADmg));

break;

}

}

}

* 1. **Описание компонентов формы и обработчиков событий её компонентов**

Наш класс программы выглядит следующим образом:

public static class Program

{

/// <summary>

/// Главная точка входа для приложения.

/// </summary>

public static Form1 MainForm;

[STAThread]

static void Main()

{

Application.EnableVisualStyles();

Application.SetCompatibleTextRenderingDefault(false);

MainForm = new Form1();

Application.Run(MainForm);

}

}

Главная форма MainForm доступна также и другим классам для взаимодействия. Вот так выглядит код обработчиков событий её компонентов:

public partial class Form1 : Form

{

public BattleField BattleF { get; private set; }

public Form1()

{

InitializeComponent();

UnitChoiceBox.Items.Add("Melee");

UnitChoiceBox.Items.Add("Ranged");

UnitChoiceBox.SelectedItem = UnitChoiceBox.Items[0];

UnitChoiceBox.SelectedIndex = 0;

FullClearBtn.Visible = false;

StartFightBtn.Visible = false;

UnitChoiceBox.Visible = false;

}

private void FullClearBtn\_Click(object sender, EventArgs e)//очищает поле целиком

{

if (BattleF != null)

BattleF.ClearFullField(true);

else

MessageBox.Show("Initialize battle field first!");

}

private void InitializeBattleFieldBtn\_Click(object sender, EventArgs e)

//создаёт поле битвы и открывает доступ к остальному интерфейсу

{

BattleF = new BattleField();

BattleF.UnitChoice = (string)UnitChoiceBox.SelectedItem;

InitializeBattleFieldBtn.Visible = false;

FullClearBtn.Visible = true;

StartFightBtn.Visible = true;

UnitChoiceBox.Visible = true;

}

private void StartFightBtn\_Click(object sender, EventArgs e)//начать битву

{

if (BattleF != null)

BattleF.StartFight();

else

MessageBox.Show("Initialize battle field first!");

}

private void UnitChoiceBox\_SelectedIndexChanged(object sender, EventArgs e)

//выбор нового вида воинов для расположения на поле

{

if (BattleF != null)

BattleF.UnitChoice = (string)UnitChoiceBox.SelectedItem;

}

public void ToggleButtons()

//метод, переключающий доступ к интерфейсу на время битвы

{

FullClearBtn.Enabled = !FullClearBtn.Enabled;

StartFightBtn.Enabled = !StartFightBtn.Enabled;

UnitChoiceBox.Enabled = !UnitChoiceBox.Enabled;

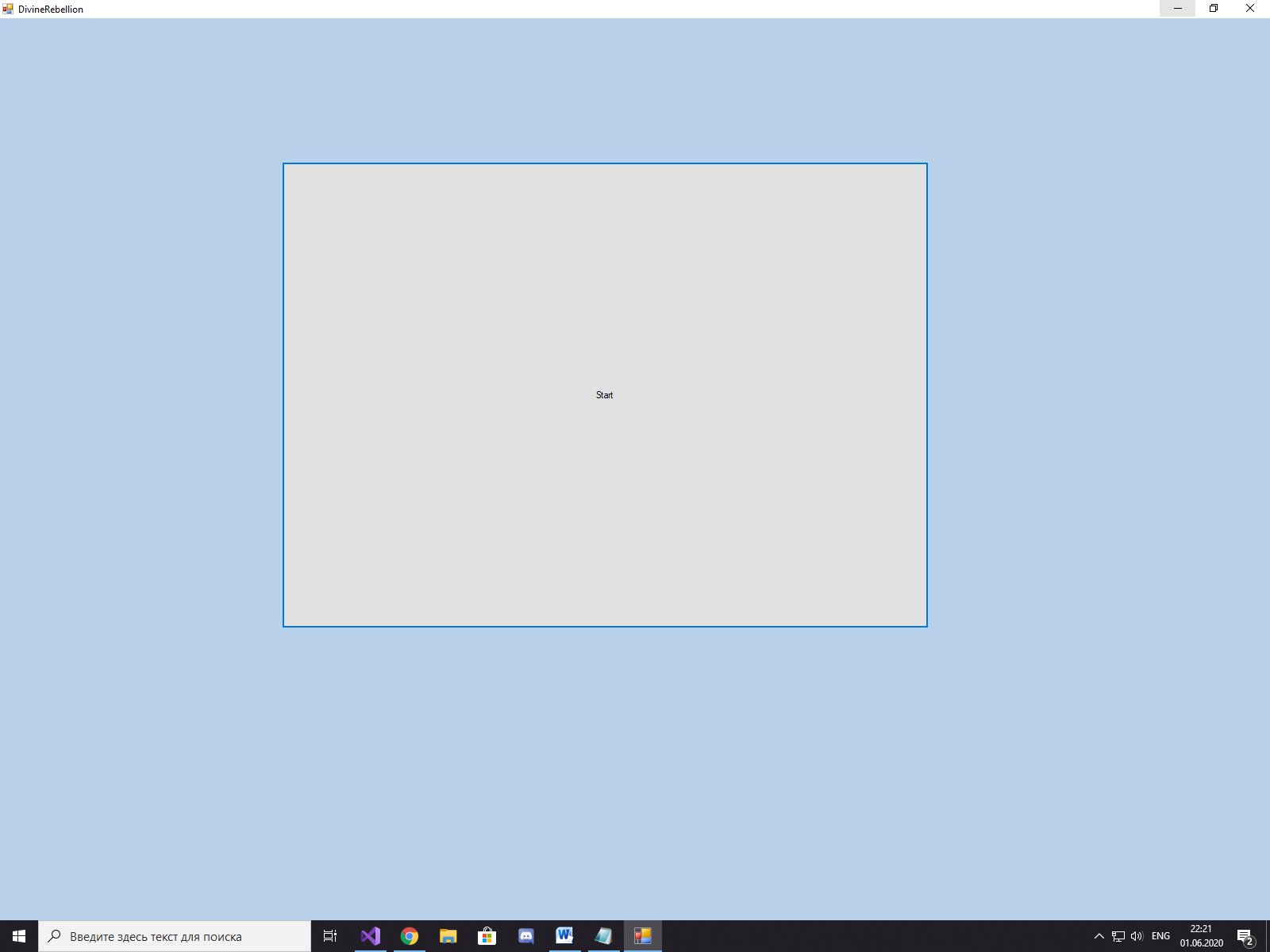
}

}

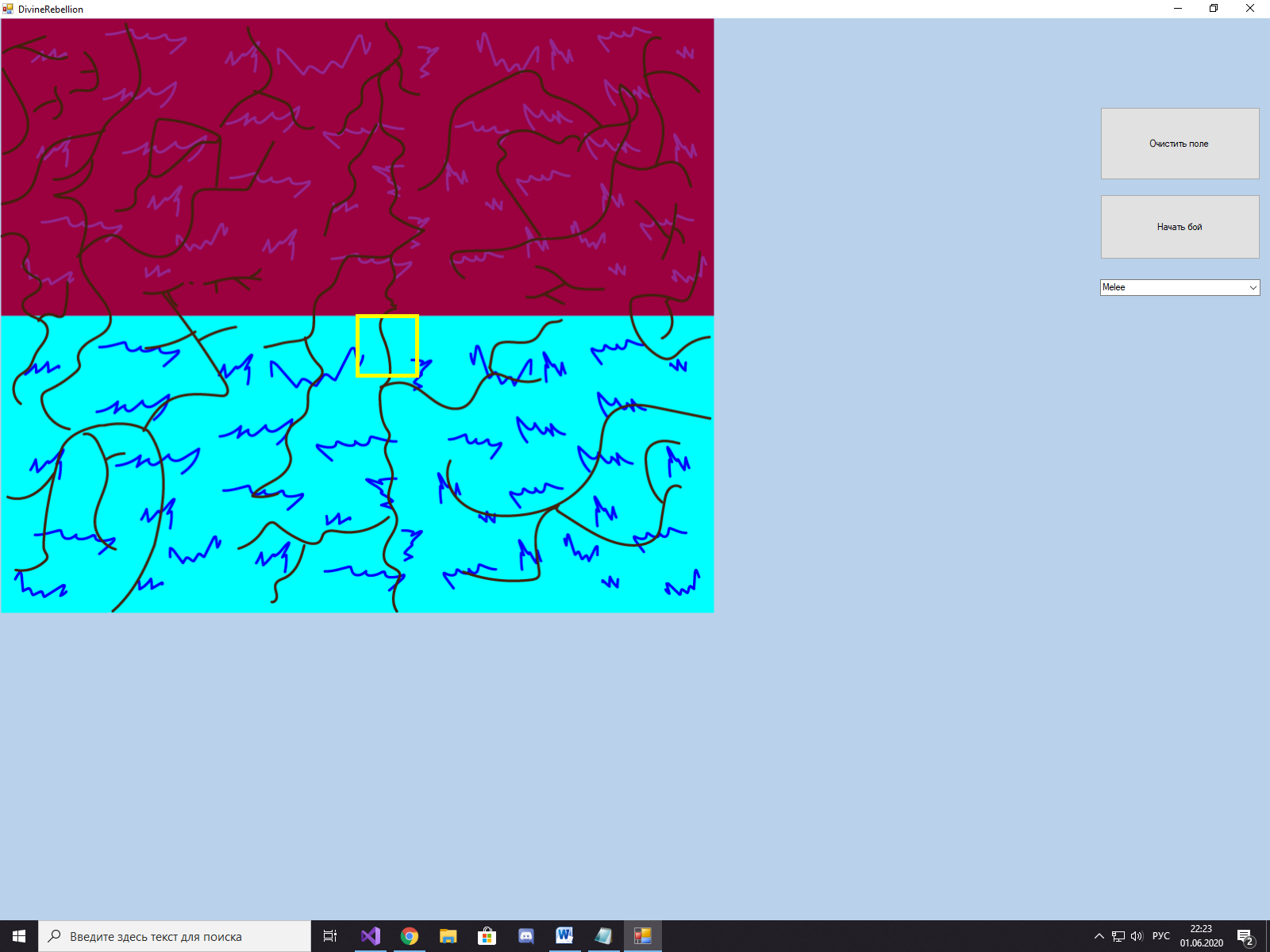
Для большего понимания предоставляем вам иллюстрации:

* [Форма при запуске](#Рис1) (рис. 1)
* [Форма после начала игры](#Рис2) (рис. 2)
* [Размещение воинов на поле](#Рис3) (рис. 3)
* [Сражение](#Рис4) (рис. 4)
* [Текстура воинов ближнего боя](#Рис5) (рис. 5)
* [Текстура воинов дальнего боя](#Рис6) (рис. 6)
* [Текстура снаряда](#Рис7) (рис. 7)
* [Текстура поля битвы](#Рис8) (рис. 8)

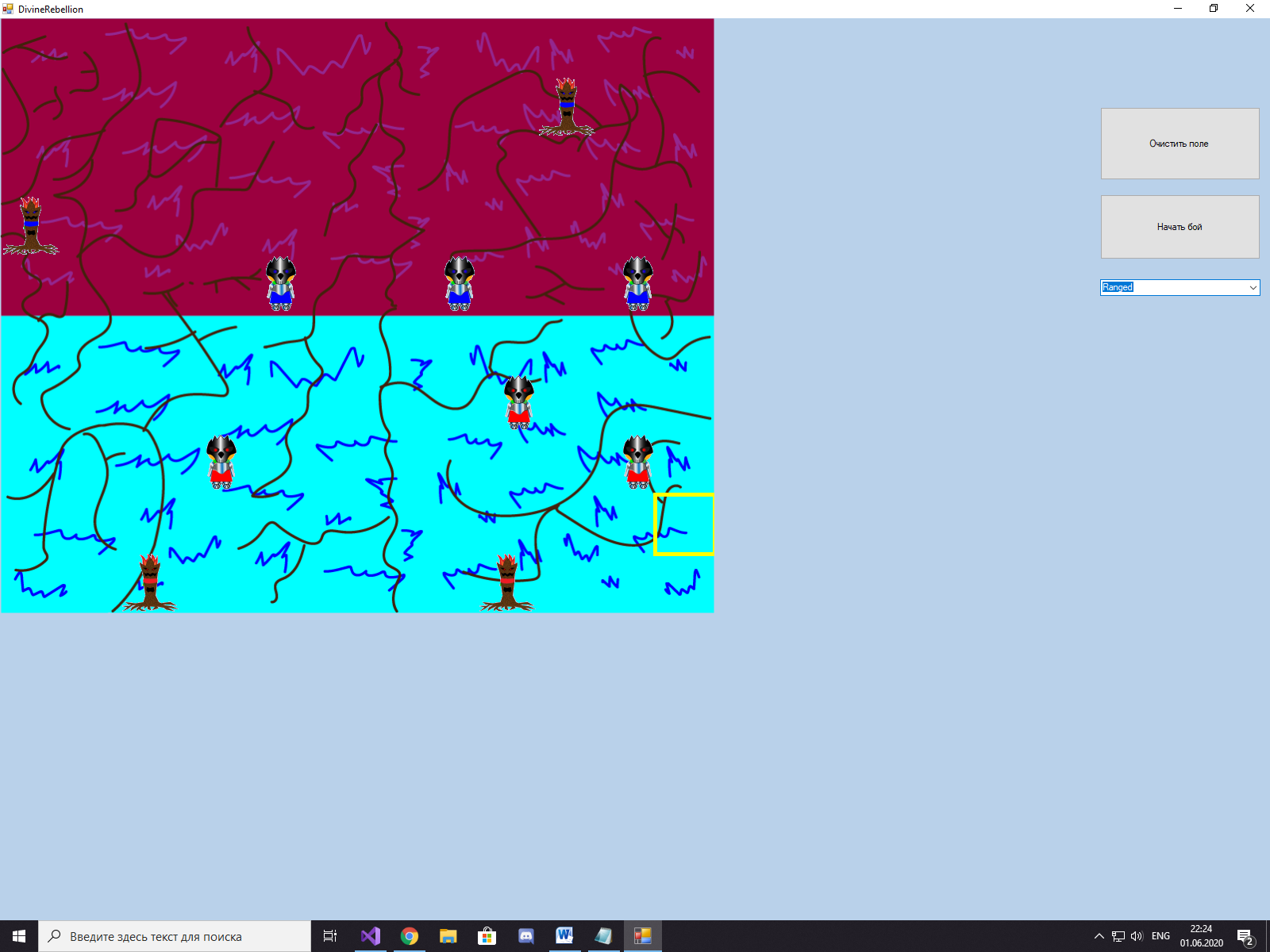
Разработка текстур производилась с помощью программы Adobe Illustrator 2020.[[3]](#И3)



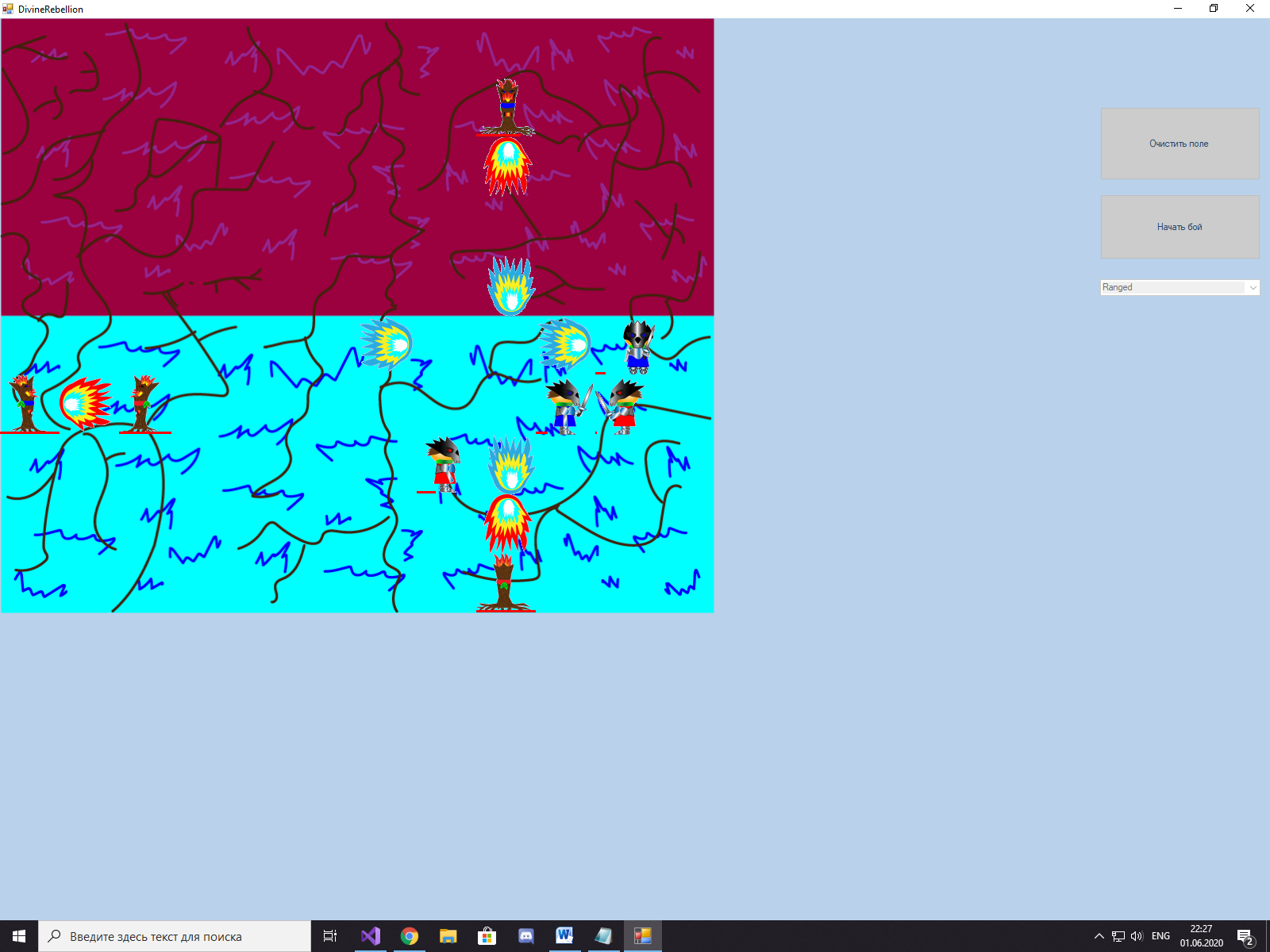
(Рис. 1) Окно формы сразу после запуска [InitializeBattleFieldBtn активен]



(Рис. 2) Окно формы после нажатия кнопки Start



(Рис. 3) Размещение воинов на поле перед боем [MouseMove, MouseClick]



(Рис. 4) Разгар сражения [BattleField.StartFight]

Класс BattleField имеет у себя следующее поле: private PictureBox FieldBox;

К нему привязаны события FieldBox.MouseMove и FieldBox.MouseClick. В конструкторе они привязаны следующим образом:

FieldBox.MouseMove += new MouseEventHandler(FieldBox\_MouseMove);

FieldBox.MouseClick += new MouseEventHandler(FieldBox\_MouseClick);

Код обработчиков данных событий вы можете видеть ниже:

private void FieldBox\_MouseMove(object sender, MouseEventArgs e)

{

if (allowMouseMove)

{

FieldBox.Image.Dispose();

FieldBox.Image = (Image)InitField.Clone();

Bitmap bmp = FieldBox.Image as Bitmap;

Graphics graphics = Graphics.FromImage(bmp);

graphics.DrawRectangle(Pen, (e.X / resolution) \* resolution, (e.Y / resolution) \* resolution, resolution, resolution);

FieldBox.Image = bmp;

graphics.Dispose();

}

}

private void FieldBox\_MouseClick(object sender, MouseEventArgs e)

{

if (allowMouseClick)

{

if (BattleTiles[e.Y / resolution, e.X / resolution].IsFree)

{

if (e.Button == MouseButtons.Left)

{

Team team;

if (e.Y < height / 2)

team = Team.Blue;

else

team = Team.Red;

switch (UnitChoice)

{

case "Melee":

BattleTiles[e.Y / resolution, e.X / resolution].Warrior = new Melee(team, e.X / resolution, e.Y / resolution);

break;

case "Ranged":

BattleTiles[e.Y / resolution, e.X / resolution].Warrior = new Ranged(team, e.X / resolution, e.Y / resolution);

break;

}

BattleTiles[e.Y / resolution, e.X / resolution].IsFree = false;

LoadTexture(BattleTiles[e.Y / resolution, e.X / resolution].Warrior.Texture, (e.X / resolution) \* resolution,

(e.Y / resolution) \* resolution, BattleTiles[e.Y / resolution, e.X / resolution].Warrior.Dir,

BattleTiles[e.Y / resolution, e.X / resolution].Warrior.IsAttacking);

}

}

else

{

if (e.Button == MouseButtons.Right)

{

BattleTiles[e.Y / resolution, e.X / resolution].Warrior.Dispose();

ClearCell((e.X / resolution) \* resolution, (e.Y / resolution) \* resolution);

BattleTiles[e.Y / resolution, e.X / resolution].Warrior = null;

BattleTiles[e.Y / resolution, e.X / resolution].IsFree = true;

}

}

}

}

Если активен флаг allowMouseMove, то когда пользователь будет водить мышкой по клиентской области FieldBox, он будет видеть выбранную клетку на поле, она будет обведена жёлтым квадратом.

Если же активен флаг allowMouseClick, то когда пользователь нажмёт на Левую клавишу мыши, находясь в клиентской области FieldBox, на пустой клетке появится выбранный в UnitChoiceComboBox воин, а на непустой клетке ничего не произойдёт. Наоборот, если пользователь нажмёт на Правую клавишу мыши, находясь в клиентской области FieldBox, то непустая клетка будет очищена от находящегося там воина, а на пустой клетке ничего не произойдёт.

Рассмотрим также событие OnDied, принадлежащее классу BattleField.

Ниже вы можете видеть его реализацию с помощью делегата EventHandler, а также реализацию пользовательского класса TileEventArgs, унаследованного от общего класса EventArgs:

public class TileEventArgs : EventArgs

{

public int X { get; set; }

public int Y { get; set; }

}

//в классе BattleField

public EventHandler<TileEventArgs> OnDied;//событие для обработки умершего воина

public BattleField()

{

//…

OnDied += FreeBCell;

//…

}

private void FreeBCell(object sender, TileEventArgs e)

{

Unit warrior = sender as Unit;

if (warrior.Equals(BattleTiles[e.Y, e.X].Warrior))

{

BattleTiles[e.Y, e.X].Warrior.Dispose();

BattleTiles[e.Y, e.X].Warrior = null;

BattleTiles[e.Y, e.X].IsFree = true;

}

}

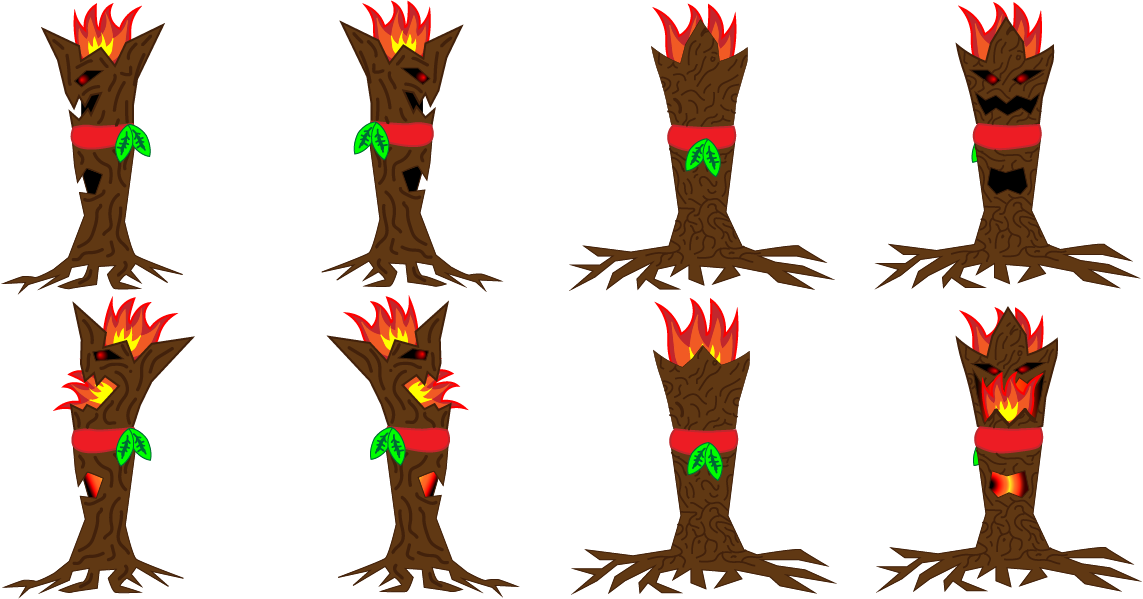
Данное событие вызывается в методе Unit.TakeDamage(…) и срабатывает, когда воин на поле получает критический урон.

* 1. **Краткое описание текущей версии программы**

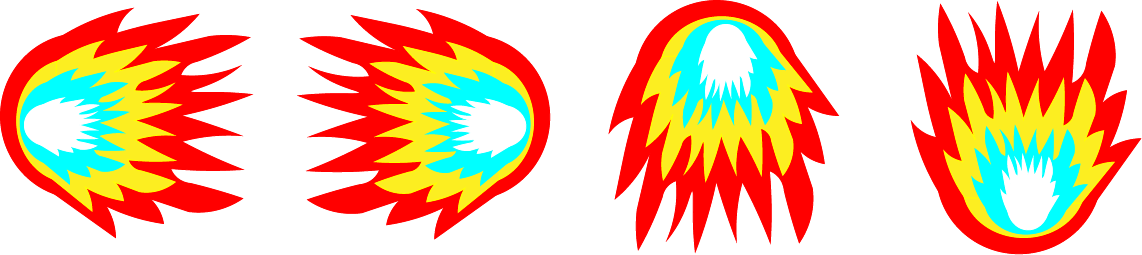
Данная версия игры представляет собой реализацию боевой системы с рабочим графическим интерфейсом и минимальным набором воинов для расстановки на поле. В бою преимущество получают воины, стоящие выше и левее оппонента, в силу специфики алгоритма хода, поэтому расстановка на поле будет играть ключевую роль в бою. Звук, однако, отсутствует. Ниже представлены текстуры, используемые для отрисовки воинов и снарядов на поле:



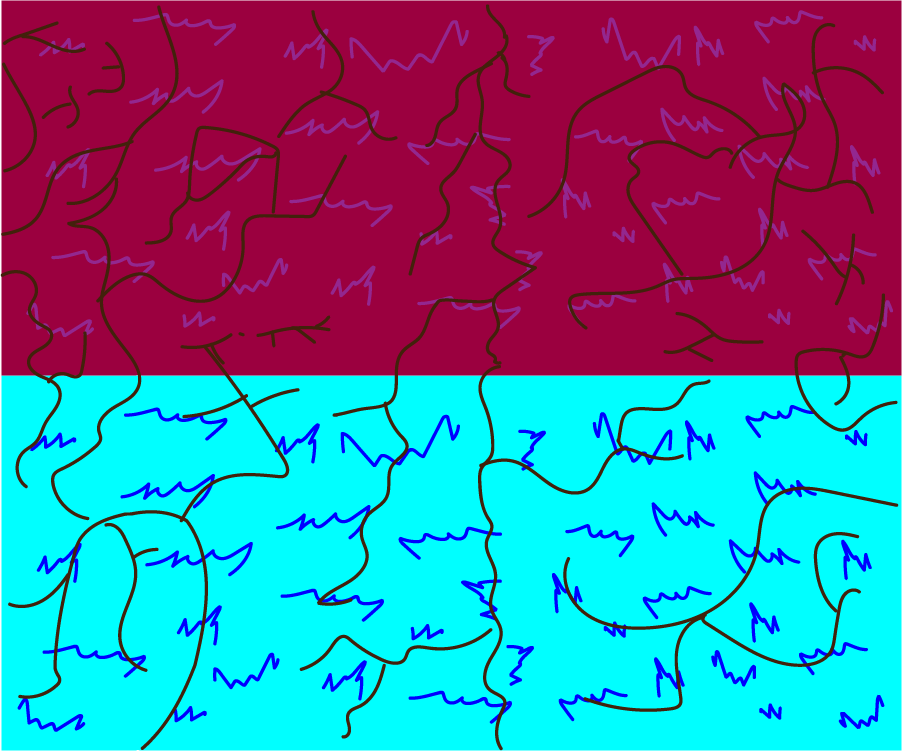
(Рис. 5) Текстура для воинов ближнего боя синей команды



(Рис. 6) Текстура для воинов дальнего боя красной команды



(Рис. 7) Текстура для снаряда, выпускаемого воином дальнего боя красной команды



(Рис. 8) Текстура поля битвы

**ЗАКЛЮЧЕНИЕ**

В результате выполнения курсовой работы была разработана первая версия игры в стиле Autochess «Divine Rebellion» с возможностью расставлять воинов на поле и проводить бои между двумя командами. В ходе его разработки были получены теоретические и практические знания по языку программирования C# и Windows Forms API. Создание новых видов воинов можно довольно легко производить на основе общего ADT Unit, а специфические особенности взаимодейтсвия с клетками на игровом поле (как в случае с Ranged) можно добавлять параллельно модифицируя класс Tile и, возможно, TimeEventProcessor (это обработчик тика таймера BattleTimer, который представляет собой ход на игровом поле) в соответствующих местах. Новые файлы текстур воинов можно привязать к их именам, а для создания их экземпляров можно будет передавать конструктору класса специальную структуру, определяющую характеристики нового воина, включая имя. В этом и будет заключаться дальнейшая разработка проекта, а на текущий момент представлена простейшая его версия.

**СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

METANIT.COM [Электронный ресурс] – <https://metanit.com>

WIKIPEDIA [Электронный ресурс] – <https://ru.wikipedia.org>

YouTube “Alex Andr” [YouTube канал] – <https://www.youtube.com/channel/UCGL1HHCewP_Fp8OhE_ai05g>

Язык программирования C# 7 и платформы .NET и .NET Core, 8-е изд., Э. Троелсен, Ф. Джепикс [часть VI, гл. 19, стр. 686 - 738]

StackOverflow [Интернет-форум] – [https://stackoverflow.com](https://stackoverflow.com/)

Cyberforum [Интернет-форум] – [https://www.cyberforum.ru](https://www.cyberforum.ru/)