«Игра с процедурно генерируемым миром»

Работу выполнил: Керченский Артур
Руководитель: Межов Александр Анатольевич

Цели и задачи

Создание игры с процедурно-генерируемым миром

Изучение литературы

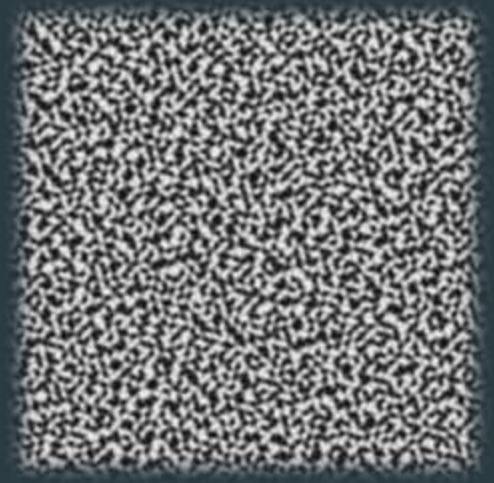
Реализация процедурной генерации

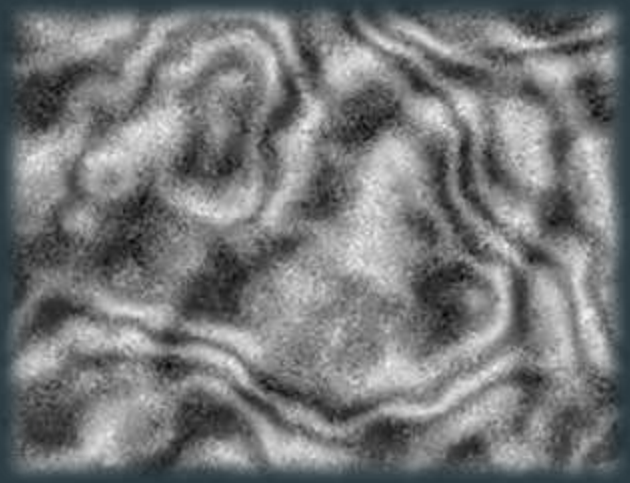
Разработка объектов игры

Оформление и презентация проекта

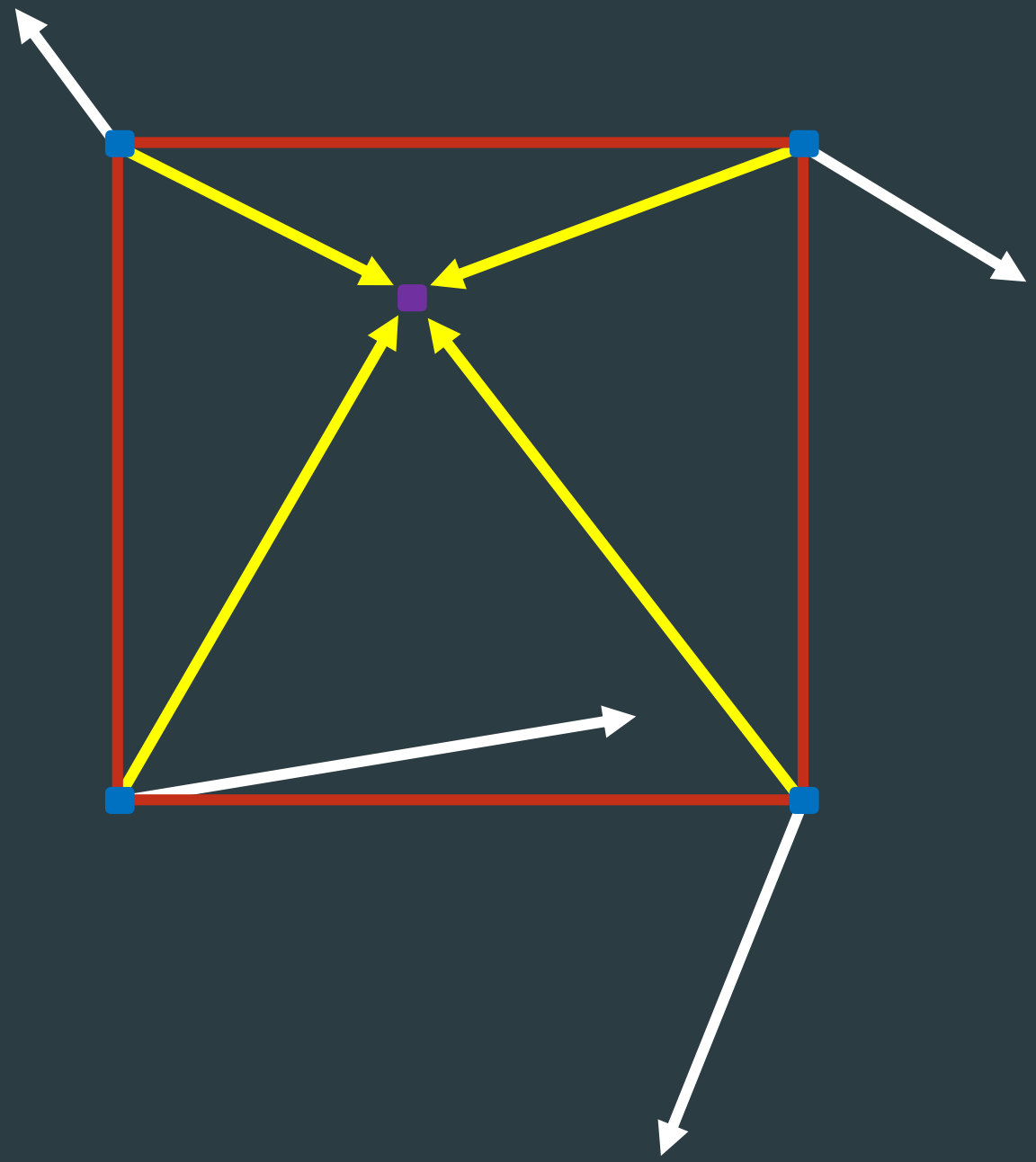
Виды процедурной генерации

Клеточные автоматы

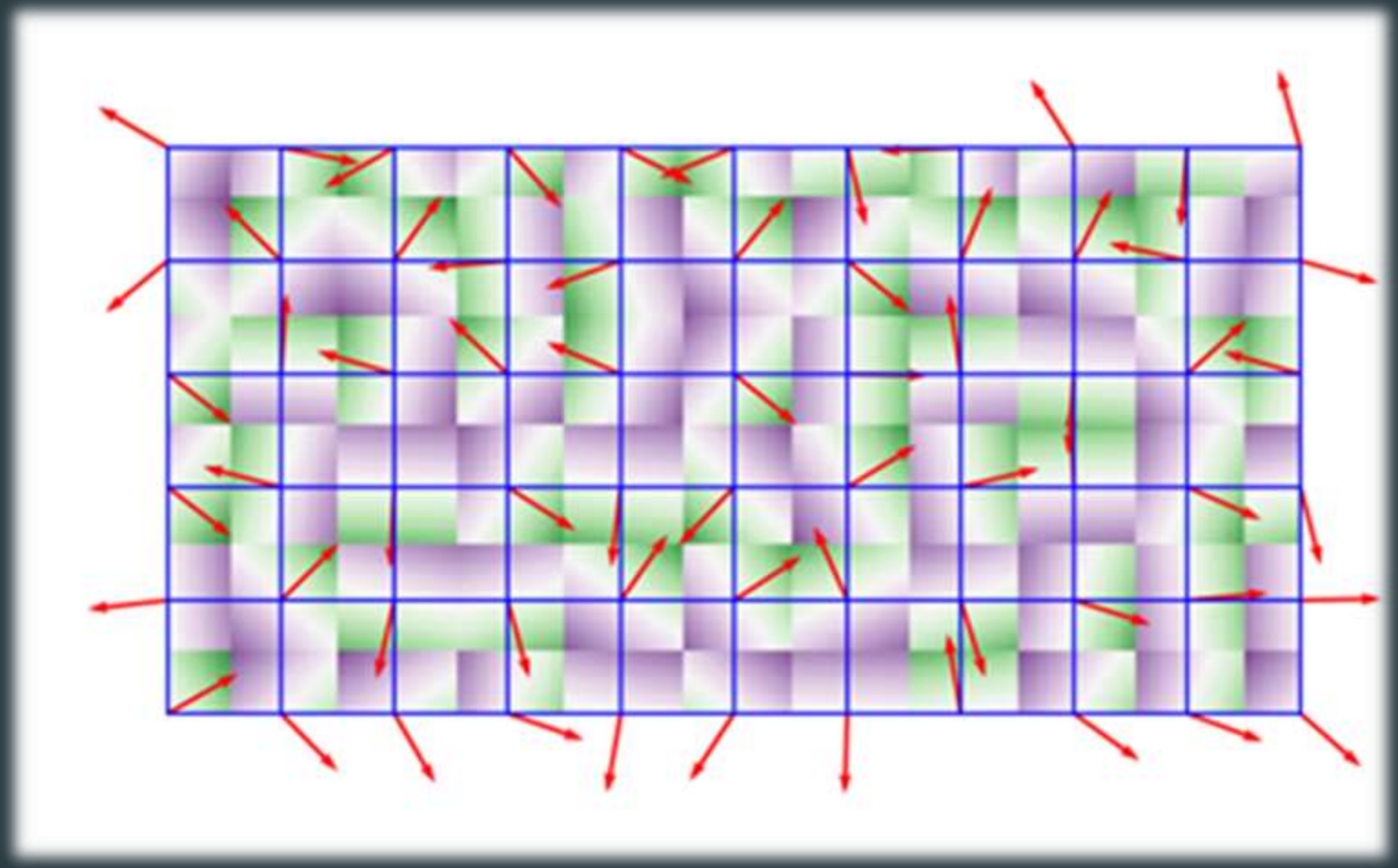




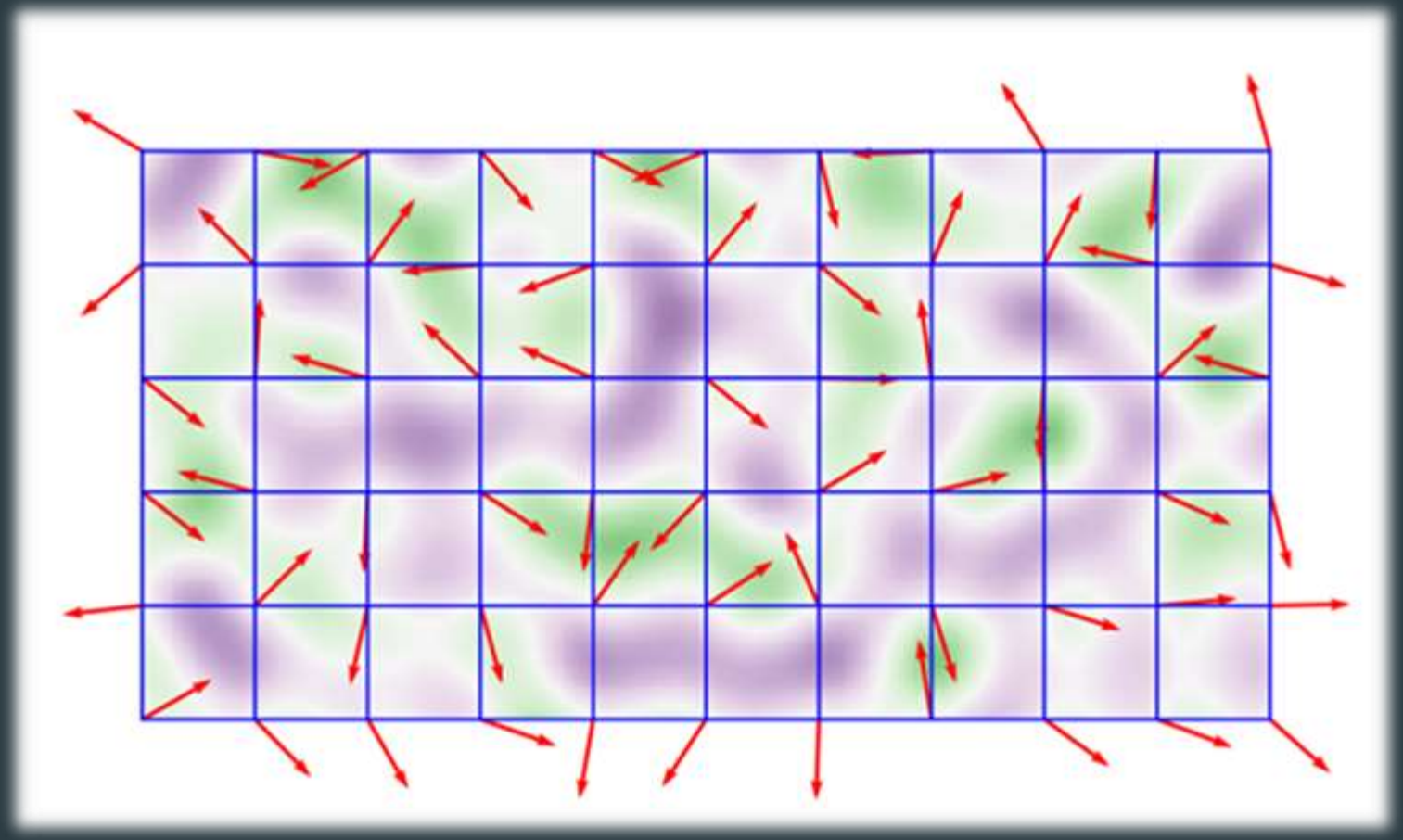
Шумы



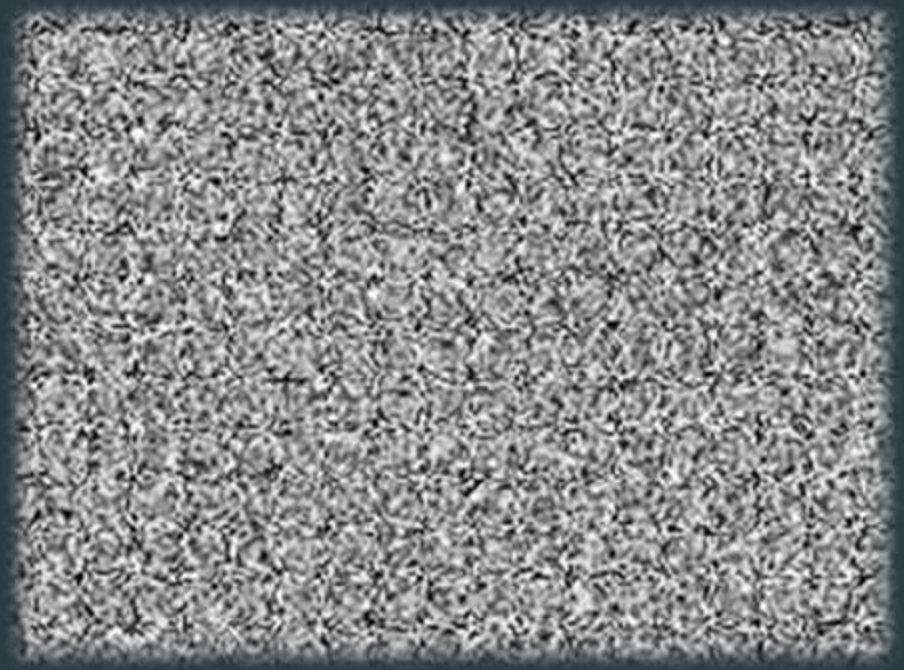
```python
import numpy as np
import matplotlib.pyplot as plot

def perlin(x, y, seed=0):
    np.random.seed(seed)
    ptable = np.arange(256, dtype=int)

    # получение свертки матрицы для линейной интерполяции
    ptable = np.stack([ptable, ptable]).flatten()

    # задание координат сетки
    xi, yi = x.astype(int), y.astype(int)

    # вычисление координат расстояний
    xg, yg = x - xi, y - yi

    # применение функции затухания к координатам расстояний
    xf, yf = fade(xg), fade(yg)

    # вычисление градиентов в заданных интервалах
    n00 = gradient(ptable[ptable[xi] + yi], xg, yg)
    n01 = gradient(ptable[ptable[xi] + yi + 1], xg, yg - 1)
    n11 = gradient(ptable[ptable[xi + 1] + yi + 1], xg - 1, yg - 1)
    n10 = gradient(ptable[ptable[xi + 1] + yi], xg - 1, yg)

    # линейная интерполяция градиентов n00, n01, n11, n10
    x1 = lerp(n00, n10, xf)
    x2 = lerp(n01, n11, xf)
    return lerp(x1, x2, yf)

def lerp(a, b, x):
    "функция линейной интерполяции"
    return a + x * (b - a)

# функция сглаживания
def fade(f):
    return 6 * f ** 5 - 15 * f ** 4 + 10 * f ** 3

# вычисление векторов градиента
def gradient(c, x, y):
    vectors = np.array([[0, 1], [0, -1], [1, 0], [-1, 0]])
    gradient_co = vectors[c % 4]
    return gradient_co[:, :, 0] * x + gradient_co[:, :, 1] * y
```



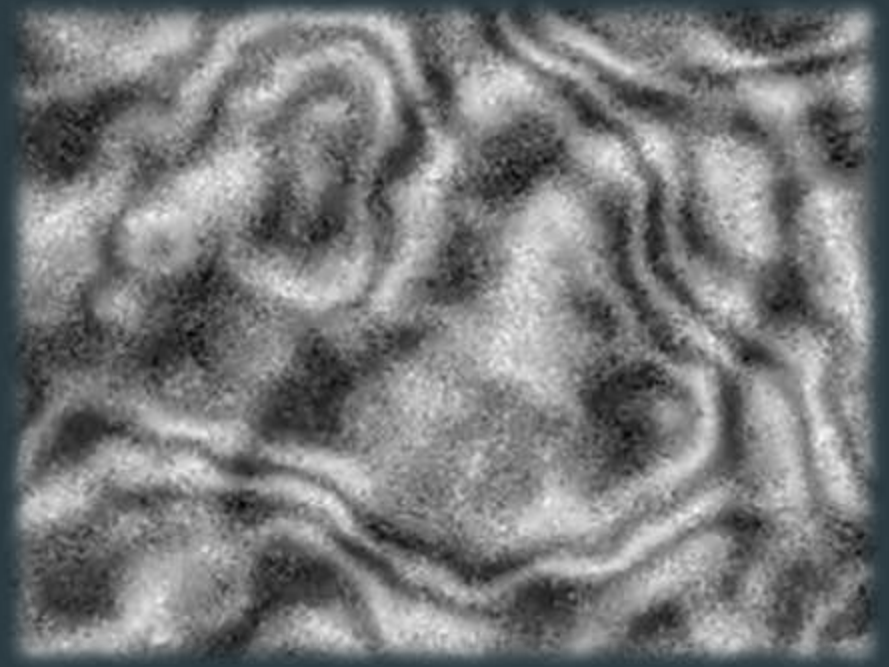
Влияние каждой точки относительно ближайшего узла сетки



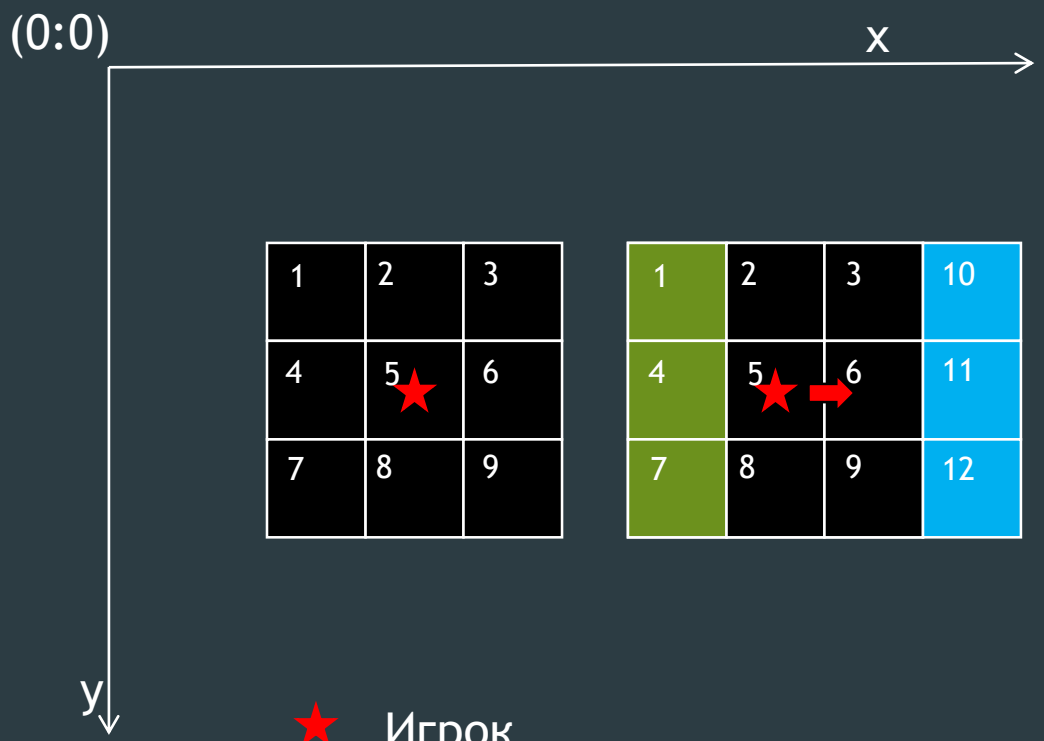
Окончательный интерполированный результат

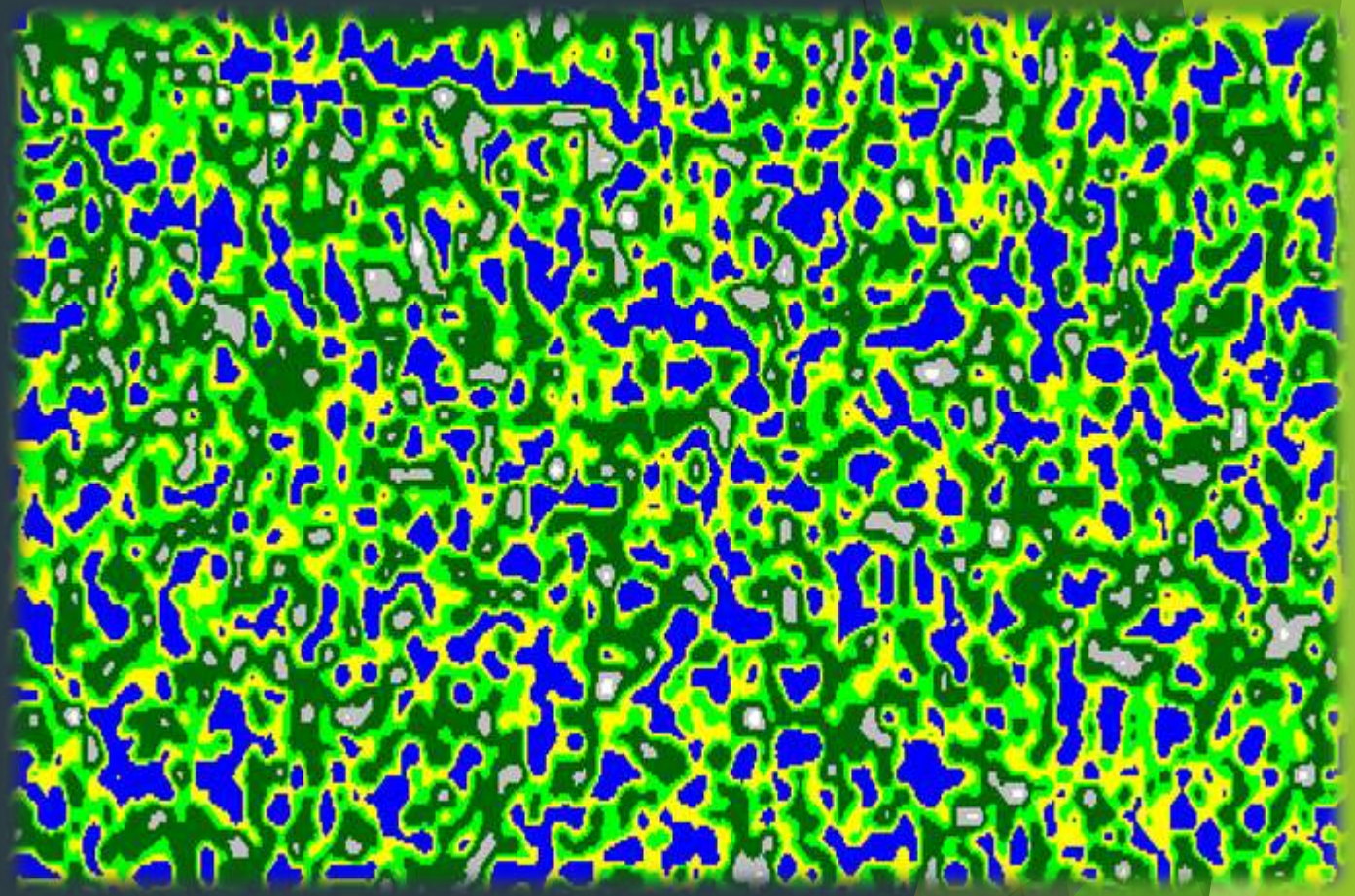


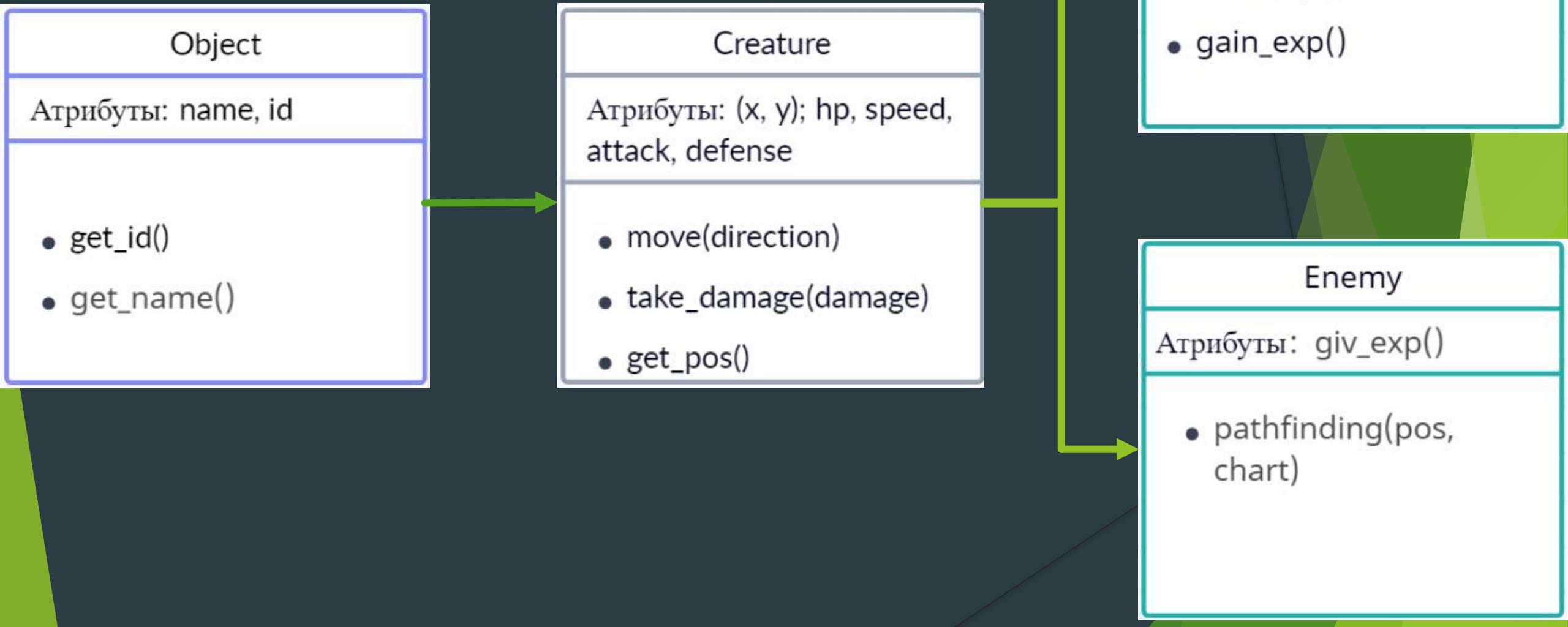
Белый шум



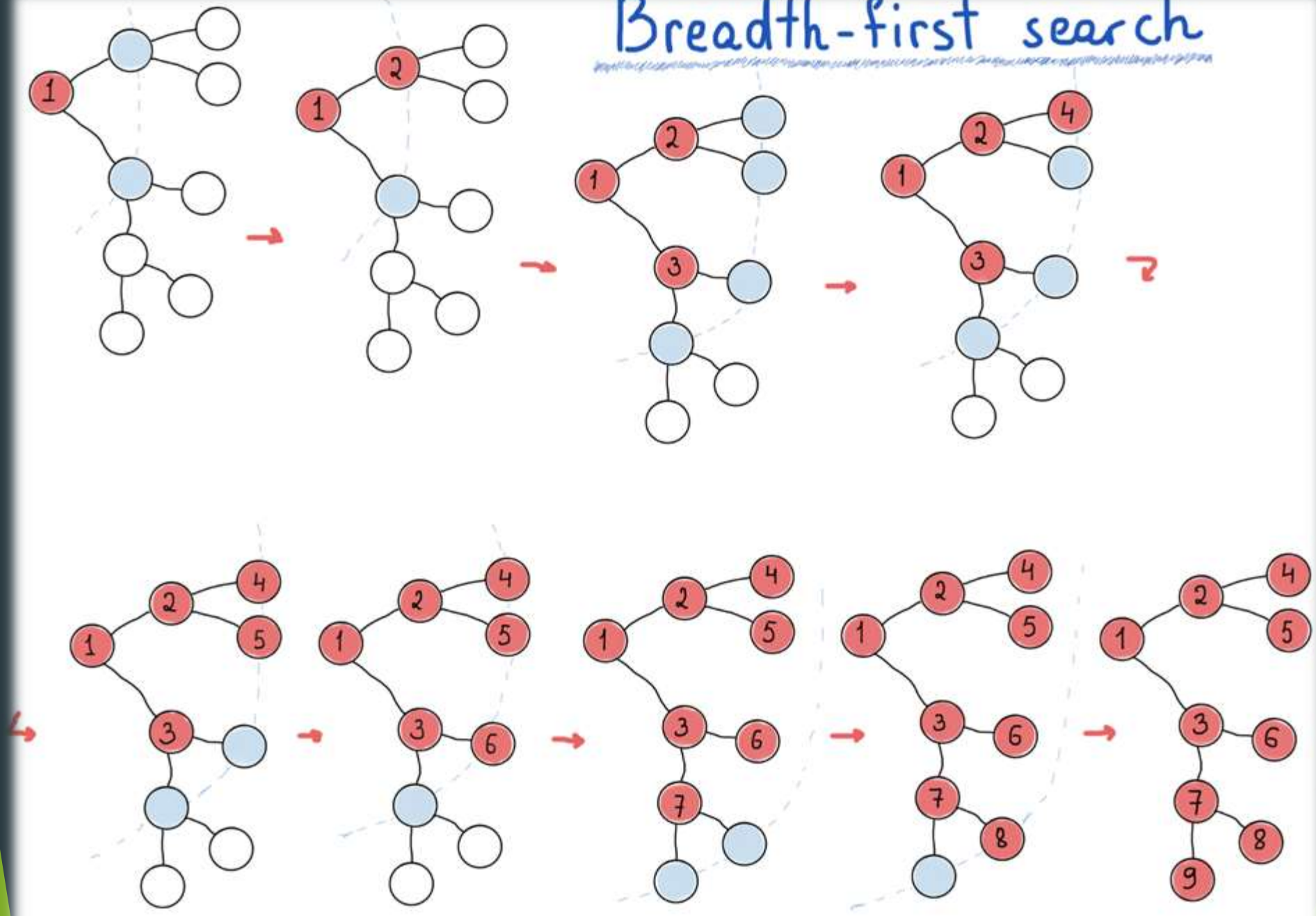
Шум Перлина


(0:0)
x
y
1
2
3
4
5
6
7
8
9
1
2
3
10
4
5
6
11
7
8
9
12
Игрок
Прогруженный чанк
Сохраненный
Созданный




Object
Атрибуты: name, id
get_id()
get_name()
Creature
Атрибуты: (x, y); hp, speed, attack, defense
move(direction)
take_damage(damage)
get_pos()
Player
Атрибуты: level, exp, power
level_up()
gain_exp()
Enemy
Атрибуты: giv_exp()
pathfinding(pos, chart)

Breadth-first search



Главное меню


```
def blitter(*args):
    components = [elem for elem in args]
    if len(components) == 7:
        screen.blit(components[0], (0, 0))
        screen.blit(components[1], (320, 168))
        screen.blit(components[2], (299, 210))
        screen.blit(components[3], (295, 253))
        screen.blit(components[4], (319, 297))
        screen.blit(components[5], (115, 70))
        screen.blit(components[5], (117, 72))
        screen.blit(components[6], (116, 71))
    else:
        screen.blit(components[0], (0, 0))
        screen.blit(components[7], (305, 160))
        screen.blit(components[1], (327, 196))
        screen.blit(components[2], (313, 230))
        screen.blit(components[3], (310, 265))
        screen.blit(components[4], (326, 300))
        screen.blit(components[5], (115, 70))
        screen.blit(components[5], (117, 72))
        screen.blit(components[6], (116, 71))
```

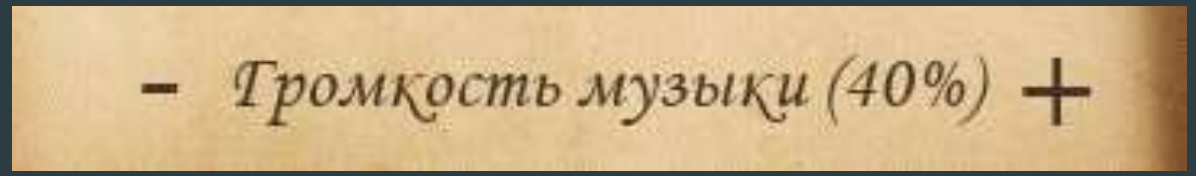
Внешний вид меню


```
def load_config(*args):
    global data
    filename = os.path.join('data/config', "cfg.txt")
    if not os.listdir("data/config"):
        config = open(filename, "w")
        config.write("resolutionWidth=1280\n")
        config.write("resolutionHeight=720\n")
        if os.listdir(path='data/saves'):
            config.write("anySaves=1\n")
        else:
            config.write("anySaves=0\n")
        config.write("difficult=1\n")
        config.write("volume=30\n")
    config = open(filename, "r+")
    raw_data = config.read()[:-1]
```

```
if args[1] == "res":
    config = open(filename, "w")
    config.write(f"resolutionWidth={args[0][0]}\n")
    config.write(f"resolutionHeight={args[0][1]}\n")
    config.write(f"anySaves={data['anySaves']}\n")
    config.write(f"difficult={data['difficult']}\n")
    config.write(f"volume={data['volume']}\n")
    config = open(filename, "r+")
```

```
if args[1] == "dif":
    config = open(filename, "w")
    config.write(f"resolutionWidth={data['resolutionWidth']}\n")
    config.write(f"resolutionHeight={data['resolutionHeight']}\n")
    config.write(f"anySaves={data['anySaves']}\n")
    config.write(f"difficult={args[0]}\n")
    config.write(f"volume={data['volume']}\n")
    config = open(filename, "r+")
```

Звуковое сопровождение



В проекте используются фоновая музыка для главного меню и звуки окружения непосредственно в самой игре. Они регулируются при помощи соответствующего пункта в разделе настроек главного меню

Параметр громкости хранится отдельно в директории с файлами, где происходит его считывание, изменение и сохранение при выходе из игры.

```
def play_music():
    data = load_config()
    pygame.mixer.music.load("data/music/menu.mp3")
    pygame.mixer.music.set_volume(data["volume"]/100)
    pygame.mixer.music.play(-1)
```

```
def play_ambient():
    data = load_config()
    ambient = choice(["ambient1.mp3", "ambient2.mp3", "ambient3.mp3", "ambient4.mp3"])
    pygame.mixer.music.load(f"data/music/{ambient}")
    pygame.mixer.music.set_volume(data["volume"]/100)
    pygame.mixer.music.play(-1)
```

Спрайты

Заключение

Мы создали игру, с процедурно-генерируемым миром. Изучили библиотеку Pygame, написали алгоритмы поиска в ширину и Шум Перлина и научились работать со спрайтами.

Будущее развитие проекта

Возможности для доработки и улучшения:

- Расширение биомов и генерации
- Добавление системы предметов, инвентаря героя
- Оптимизация кода
- Внедрение возможности строительства
- Создание многопользовательских режимов игры
- Разработка системы сохранений

Спасибо за внимание