СОДЕРЖАНИЕ

1 О программе

Является программой для симуляции естественного отбора с графическим интерфейсом.

1.1 Функционал

В программе 4 сцены:

1) Стартовый экран;
— Перейти на экран создания мира;
— Перейти на экран выбора мира;
2) Экран настройки и создания мира
— Вернуться на стартовый экран;
— Задать стартовые параметры мира:
    — коэффициенты;
    — размер мира;
    — сид;
    — количество существ, которые будут изначально;
    — награды позиции, на которых существо будет получать/терять энергию;
    — отключить/включить возможность размножаться существам;
    — сделать мир циклическим и задать длину цикла, в новой итерации цикла геном существ основываться на геноме выживших существ из конца предыдущей итерации;
— Перейти к сцене симуляции мира;
3) Экран симуляции мира
— вернуться на стартовый экран;
— поставить на паузу/продолжить симуляцию;
— сделать 1 шаг симуляции;
— включить/выключить визуализацию симуляции мира;
— включить/выключить визуализацию наград;
— задать имя миру и сохранить мир с ним;
— посмотреть fps, количество сделанных шагов симуляции, количество живых и мёртвых существ;
4) Экрана выбора мира из сохранённых
— вернуться на стартовый экран;
— выбрать мир из сохранённых и перейти к сцене симуляции мира;
— удалить любой из существующих миров;
— некоторая часть программы использует многопоточность;
— детерминизм(при одинаковых входных данных, симуляция мира будет происходить по абсолютно одному и тому же сценарию)
— При загрузке мира его пресет(все настраиваемые параметры при создании мира) сохраняется и при создании мира нового мира будет стоять по умолчанию;

2 Инструкция для пользователя

Как пользоваться программой и проверить её свойства.

2.1 Базовые операции

2.1.1 Создание мира

Для создания мира надо нажать кнопку «Создать новый мир».
Вы перейдёте к настройке мира, после чего надо пролистать в самый низ, снова нажать кнопку «Создать новый мир».
Вы перейдёте к симуляции мира, и чтобы увидеть симуляцию в процессе достаточно убрать галочку с «Пауза».

Вы увидите, что-то похожее на это.

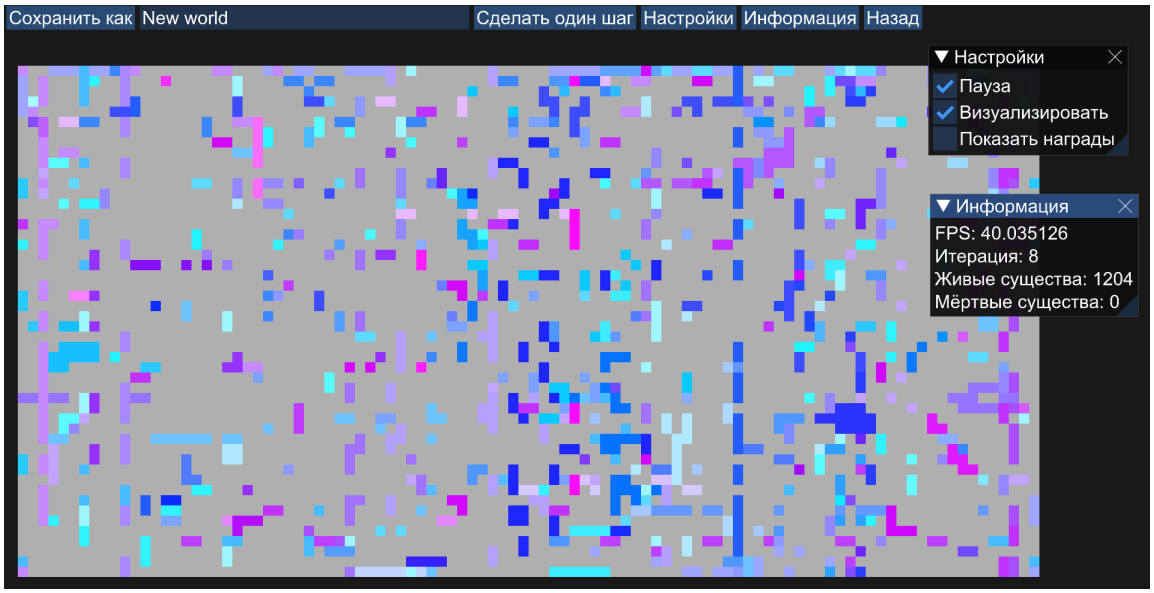

Рисунок 2.1 — Симуляция мира

2.1.2 Настройка мира

При переходе на сцену настройки мира вас встретит экран.

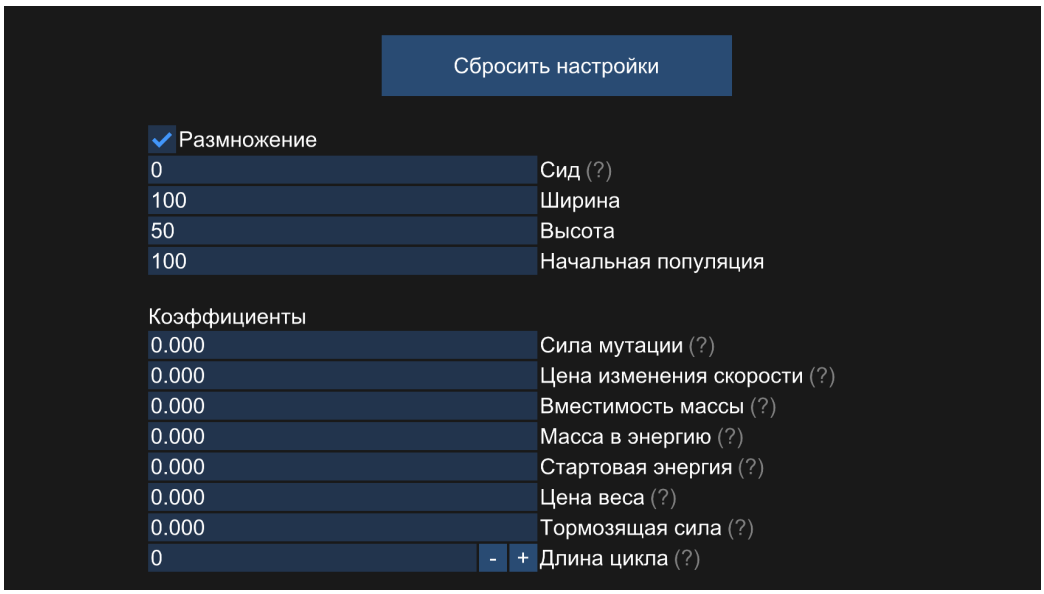

Рисунок 2.2 — Экран настройки мира(1-ая половина)

Если ширина или длина мира становятся слишком большой, то они уменьшаются до такого размера, чтобы всё могло поместиться на экран.

Если при создании мира сид =0, то он меняется на случайное значение.

Понять на что влияют коэффициенты можно наведя курсор мыши на знак вопроса в самой программе или см(подраздел 4.1.2). В базовом пресете энергия не играет никакой роли, поэтому самая эффективная стратегия - размножаться каждый ход. Меняйте коэффициенты, чтобы менять оптимальную стратегию существа, для большинства коэффициентов предполагалось, что они могут быть отрицательными.

«Длина цикла» - После x шагов(x - длина цикла), мир оказывается в начальном состоянии, однако геном новых существ базируется на геноме существ, выживших в конце предыдущей итерации.

Ниже располагается полотно, с помощью которого можно задать позиции, за нахождение на которых живое существо будет получать энергию, в количестве «Сила награды».

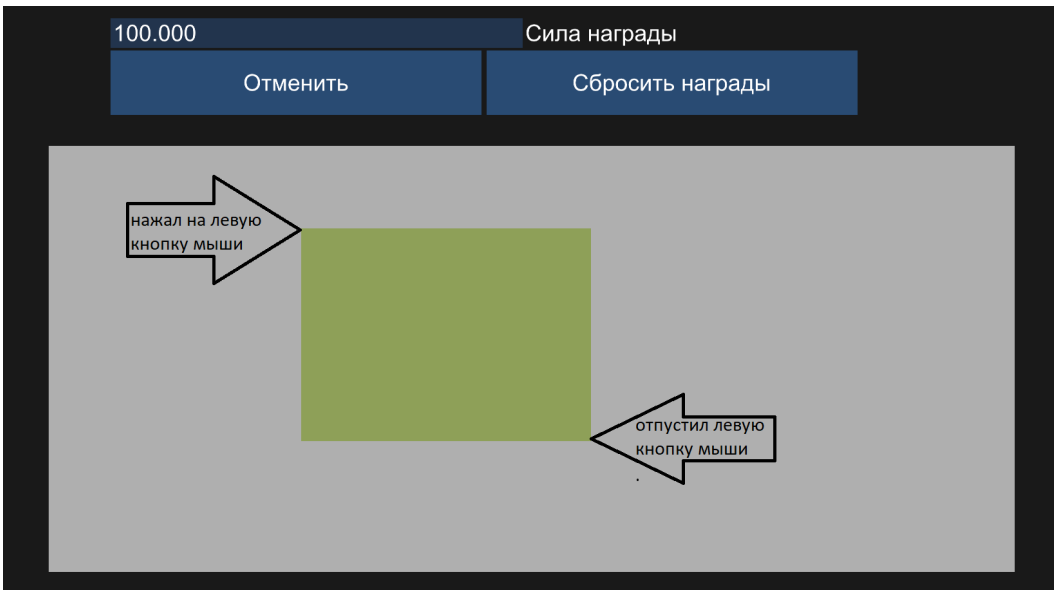

Рисунок 2.3 — Экран настройки мира(2-ая половина)

2.1.3 Что вообще происходит во время симуляции

На 1 позиции может не быть существа или 1 живое существо или 1 мёртвое. Живое существо имеет ген, думает, может тратить энергию на передвижение и размножение. Ген определяет цвет, вес и поведение существа.

Мёртвое существо имеет энергию, передвигается только по инерции, то есть если живое существо с ненулевой скоростью умирает, то оно продолжает движение будучи мёртвым.

У живого существа цвет определяется от генома, его цвет всегда имеет максимальный оттенок синего.

У мёртвого существа цвет зависит от энергии, однако его цвет полностью лишён синего оттенка.

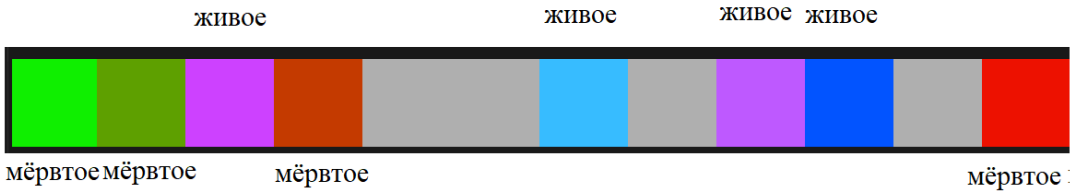

Рисунок 2.4 — Отличие между живыми и мёртвыми существами по цвету

Рисунок 2.5 — Зависимость цвета от энергии мёртвого существа

Если у живого существа энергия становится меньше 0, то оно умирает, при этом масса существа преобразуется в энергию.

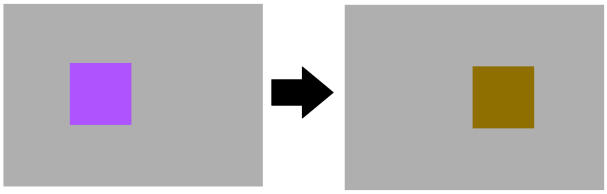

Рисунок 2.6 — Существо умерло

Если у живого существа энергия>макс энергия, то при смене позиции его энергия становится равной макс энергии, а излишки остаются на предыдущей позиции в виде мёртвого существа.

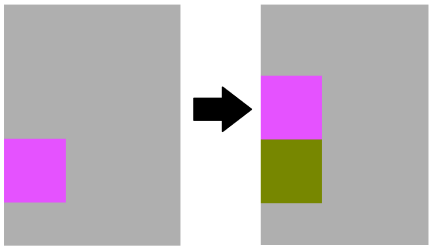

Рисунок 2.7 — Существо избавилось от лишней энергии

За 1 ход живое существо может изменить скорость, направление скорости(бесплатно) и размножиться.

2.1.4 Сохранение, загрузка, удаление мира

Чтобы сохранить мир, сначала надо его создать, затем можно дать имя миру или оставить «New world» и нажать кнопку «Сохранить как». При сохранении мира не рекомендуется использовать кириллицу.

Чтобы загрузить или удалить мир надо на стартовом экране нажать кнопку «Загрузить мир», выбрать подходящий мир и нажать соответствующую кнопку.

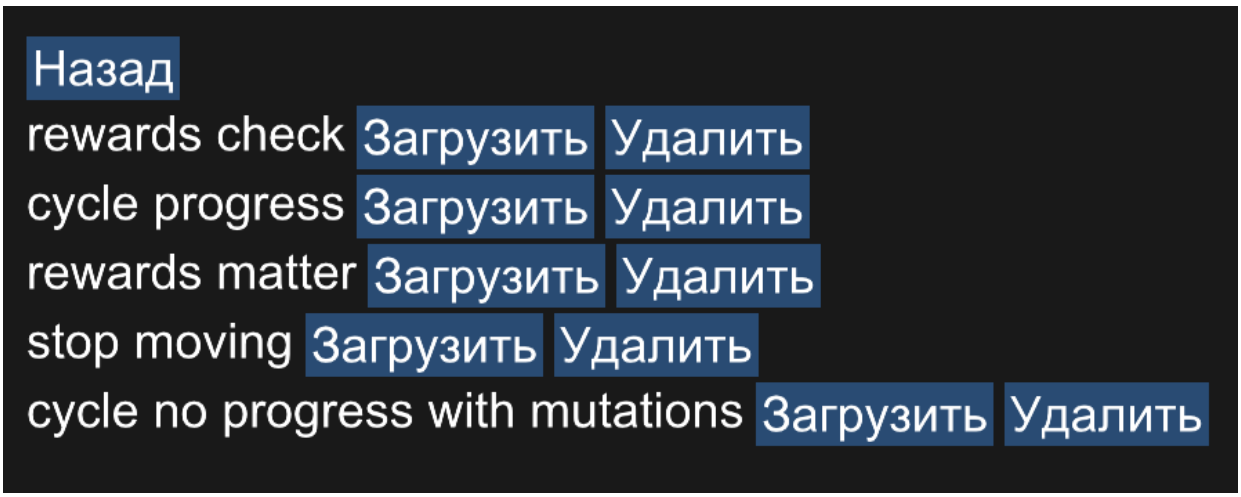

Рисунок 2.8 — Экран выбора мира с предустановленными мирами

2.2 Предустановленные миры

Демонстрируют возможности симуляции(циклы, награды, коэффициенты).

1) rewards check - сделайте 2 шага, существа, расположенные на позициях с положительной наградой, высвободят лишнюю энергию, существа, расположенные на позициях с отрицательной наградой, умрут. Тем самым, участки с ненулевыми наградами станут соответствующего награде цвета, что можно проверить с помощью кнопки «Показать награды».

2) cycle progress - циклический мир, в котором единственный источник энергии для существ - движение, в первых циклах конечное количество живых существ будет меньше начального, так как геном существ нового цикла основывается на геноме выживших предыдущего цикла, то количество выживших будет расти(состояние симуляции относительно стабилизируется на 1800 итерации).

3) rewards matter - циклический мир, в котором единственный источник энергии для существ - награды с положительной энергией на правой половине экрана.

4) stop moving - циклический мир с отключённым размножением, любое движение - смерть, количество выживших растёт с 30

5) cycle no progress with mutations - такой же мир как cycle progress, но коэффициент ”Сила мутации”не равен 0 и мир загружается сразу на 3000 итерации. Демонстрирует, что если мутация не отключена, то в циклическом мире прогресса не происходит.

2.3 Тестирование свойств

Чтобы протестировать многопоточность сделайте размер мира 500x250, запустите симуляцию и выключите паузу. Нажмите на кнопку «Информация», чтобы смотреть на FPS. Скорее всего он упадёт с максимальных 40. Запустите диспетчер задач, если нагрузка процессор около 100

Чтобы протестировать детерминизм запустите стандартный мир с длинной цикла равной 10. Нажмите на кнопку «Информация», чтобы следить за итерациями и количеством живых и мёртвых существ.

Не выключайте паузу. Вместо этого нажимайте на кнопку «Сделать один шаг».

Сохраните мир на итерациях 0, 5, 15 и 25.

Запомните количество живых и мёртвых существ на 25 итерации.

Выйдите из мира.

Дойдите до 25 итерации на всех только что сохранённых мирах и сравните количество живых и мёртвых существ с изначальным.

Так как пресет этого мира сохранился, то можно снова запустить мир через «Создать новый мир» ничего не меняя и снова дойти до 25 итерации.

Если во все разы количество живых и мёртвых существ на 25 итерации было одинаково, значит детерминизм работает.

3 Установка программы

Инструкцию того, как скачать исходный код, произвести сборку и инсталяцию.

3.1 Предварительная подготовка

Перед установкой проекта обязательно требуется компилятор поддерживающий C++20(автор пользуется MSVC 17), а также vcpkg и Cmake версии 3.18 или новее. Если компилятор не поддерживает OpenMP, то проект будет работать в однопоточном режиме. Для vcpkg требуется установить библиотеки с помощью данных команд:

```
vcpkg install glad[gl-api-33]:x64-windows
vcpkg install glfw3:x64-windows
vcpkg install imgui[core,glfw-binding,opengl3-binding]:x64-windows
vcpkg install eigen3:x64-windows
```

По желанию можно установить doxygen, чтобы дополнительно сгенерировать документацию.

3.2 Установка

Скачайте репозиторий по ссылке : https://github.com/m1n1tea/beautiful-spring.
Перейдите в папку исходников проекта через командную строку с помощью команды:

```
cd [путь до проекта]\makurov_m_p
```

Для сборки проекта введите:

```
cmake -S . -B ./build -DCMAKE_TOOLCHAIN_FILE=
[путь до vcpkg]/scripts/buildsystems/vcpkg.cmake

cmake --build ./build --config Release
```

После чего можно установить проект с помощью команды:

```
cmake --install ./build --config release --prefix
[путь куда хотите установить проект]
```

4 Документация

Сделана с помощью Doxygen. Не является полноценной документацией.

4.1 Перечисления

4.1.1 Нейросеть

InputNeurons

enum **cellworld::InputNeurons**

Основные входные нейроны существа Содержит информацию о внутреннем состоянии существа

Элементы перечислений

pos_x	Координата X.
pos_y	Координата Y.
speed_module	Модуль скорости
energy	Энергия
bias	Смещающий нейрон
input_neurons_count	Количество входных нейронов

LookInput

enum **cellworld::LookInput**

Содержит информацию об внешнем окружении существа. Для каждого из 4 направлений(верх,низ,право,лево) существо имеет отдельный набор нейронов. Поэтому общее количество входных нейронов равно 4 ∗ look↩ _input_count + input_neurons_count.

Элементы перечислений

distance	расстояние от существа до ближайшего существа в одном из направлений
color_red	уровень красного цвета ближайшего существа
color_green	уровень зелёного цвета ближайшего существа
color_blue	уровень синего цвета ближайшего существа
look_input_count	Количество входных нейронов

OutputNeurons

enum **cellworld::OutputNeurons**

Выходные нейроны существа

Элементы перечислений

change_speed_module	Изменить модуль скорости
vertical_or_horizontal	По какой прямой перемещается существо, вертикальной или горизонтальной
decrease_or_increase	В какую сторону перемещается существо, по увеличению или уменьшению координаты
reproduce	Желание существа размножиться
output_neurons_count	Количество выходных нейронов

4.1.2 Коэффициенты

enum **cellworld::Coefficients**

Набор коэффициентов, которые определяют правила мира.

Элементы перечислений

mass_into_energy	Коэффициент преобразования массы в энергию. Сколько энергии требуется для создания единица массы. Используется при: — смерти сущности, энергия трупа+= масса∗mass_into_energy. — рождении сущности, энергия родителя-=масса ребёнка∗mass_into_energy.
mass_capacity	Коэффициент вместимости массы. Сколько энергии может хранить в себе единица массы. Макс энергия = mass_capacity ∗ масса существа.
starting_energy	Коэффициент начальной энергии. Определяет сколько энергии имеет существо при спавне(не рождении). Энергия = Макс энергия ∗ starting_energy.
mass_cost	Коэффициент стоимости массы. Определяет сколько энергии живое существо тратит за ход на поддержание единицы массы. Энергия-=масса∗mass_cost.
change_speed_module_cost	Коэффициент стоимости изменения модуля скорости. Определяет сколько энергии существо тратит за ход на изменение модуля скорости. Энергия-=(изменение модуля скорости)^2∗change_speed_module_cost.

Элементы перечислений

braking_force	Коэффициент силы торможения. Определяет на сколько уменьшается модуль скорости в конце каждого хода. Модуль скорости-=braking_force.
mutation_strength	Коэффициент мутации. Определяет на сколько сильно будет отличатся геном ребёнка от родителя. — <=0 - копия родителя. — >=1 - геном не зависит от родителя совсем. Ген ребёнка=ген родителя∗(1-mutation_strength) + случайный ген∗mutation_strength.
coefficients_count	количество коэффициентов

4.2 Классы

4.2.1 Класс cellworld::Creature

Класс существа.

Открытые члены

- **Creature** (const **Creature** &)=default
- **Creature** (**Creature** &&)=default
- **Creature** & **operator=** (const **Creature** &)=default
- **Creature** & **operator=** (**Creature** &&)=default
- float **Leftover** ()

  Возвращает избыток энергии существа. Энергия уменьшается до лимита энергии.
- bool **wantToReproduce** () const

  проверка на желание размножиться
- void **look** (**Creature** &found, int direction)

  Получение информации о соседнем существе в одном из направлений. Для большей информации
- void **getInfo** ()

  Получение информации о себе. Для большей информации
- void **reverseInput** ()

  Нормализация входных данных. Переводит в диапозон [0;1] путём быстрого, но неточного деления на само себя.
- void **think** ()

  Работа нейросети. Обрабатывает входные нейроны с помощью весов, расположенных в геноме существа, и выводит результат в выходные нейроны.
- void **act** ()

Работа с выходными нейронами. Если существо живое, то в зависимости от выходных нейронов меняет модуль скорости и направление. После чего обновляется положение существа. В конце функции модуль скорости уменьшается на коэффициент торможения.

— void **eat** (**Creature** &)

Существо забирает всю энергию другого существа.

— void **addEnergy** (const float & **energy**)

Существо получает энергию в размере energy.

— void **buildIO** ()

Инициализирует недостающие элементы существа при чтении из файла.

Специальные конструкторы существа

По умолчанию модуль скорости равен 0, существо направлено вверх.

— **Creature** (const **Genome** &gen, const Position &pos)

Конструируется живое существо.

— **Creature** ()

Конструируется несуществующее существо

— **Creature** (float **energy**, const Position &pos)

Конструируется неживое существо.

Изменение состояния существа

Используются для избежания лишней инициализации

— void **makeAlive** (**Creature** &ancestor, const Position &pos)

Существо становится живым. Геном существа основывается на предке.

— void **makeAlive** (const Position &pos)

Существо становится живым. Геном рандомный.

— void **die** ()

Существо становится мёртвым.

— void **stopExisting** ()

Существо перестаёт существовать.

Геттеры

— const int & **getState** () const
— const int & **getDirection** () const
— const int & **getSpeed** () const

— int & **getX** ()

Возвращает ссылку на изменяемое значение, равное, но не являющееся координатой X.

— int & **getY** ()

Возвращает ссылку на изменяемое значение, равное, но не являющееся координатой Y.

— const float & **getEnergy** () const
— const float & **getEnergyLimit** () const
— const int & **getMass** () const
— const unsigned int & **getColor** () const

- unsigned int **getBlue** () const
- unsigned int **getGreen** () const
- unsigned int **getRed** () const
- const **Genome** & **getGenome** ()

Открытые статические члены

- static **Genome generateGenome** ()

  Создать случайным геном

- static void **generateGenome** (**Genome** &)

  Сделать данный геном случайным

- static void **mixGen** (float &gen1, const float &gen2)

  Смешивание генов

- static unsigned int **mixGen** (const unsigned int &gen1, const unsigned int &gen2)

  Смешивание генов

- static **Genome createGenome** (const **Genome** &ancestor)

  Создать геном, основанный на геноме предка.

- static unsigned int **energyColor** (int **energy**)

  Цвет энергии. Цвет мёртвого существа зависит от его текущей энергии.

Открытые атрибуты

Позиция существа

- int **pos_x_**
- int **pos_y_**

4.2.2 Класс cellworld::Field

Класс Поля, хранит существ.

Открытые члены

- **Field** (int size_x=0, int size_y=0)

  Конструктор класса, задающий ширину и длину поля.

- void **updatePositions** ()

  Обновляет позиции существ. Ввод данных в нейросеть. Происходит вся работа нейросетей. Забирает энергии у существ за передвижение. Оставляет на предыдущей позиции остатки энергии. Обрабатывает коллизию существ. Не инициализирует объекты.

- void **updateStates** ()

  Обновляет состояние существ. Если у живого существа $<$0 энергии - оно умирает. Если существо хочет размножаться, то одно из соседних существ становится живым, с геном, наследуемым от предка. Работает в многопоточном режиме, без инициализаций.

- Position **findClosePosition** (**Creature** $*$ancestor)

Соседняя позиция относительно существа, желающего размножиться, которая становится живой.

Если позиция не найдена, возвращает bad_position.

— **Creature** & **findCreature** (**Creature** ∗finder, int direction)

Поиск существа из определённого существа в определённом направлении.

— void **clear** ()

Очистка поля

Доступ к существу по 1 или 2 индексам(позиции) поля

- const **Creature** & **getCreature** (const Position &pos) const
- **Creature** & **getCreature** (const Position &pos)
- const **Creature** & **getCreature** (const int &index) const
- **Creature** & **getCreature** (const int &index)
- const **Creature** & **operator[ ]** (const int &index) const
- **Creature** & **operator[ ]** (const int &index)

Геттеры

- unsigned int **getColor** (const Position &pos) const
- unsigned int **getColor** (const int &index) const
- int **sizeX** () const
- int **sizeY** () const
- int **size** () const
- const int & **getAlive** ()
- const int & **getDead** ()

Проверка координаты на валидность

- bool **validX** (const int &x) const
- bool **validY** (const int &y) const

Интерфейс визуализации

- void **createTexture** ()
- void ∗ **getTexture** ()

  Получить текстуру для работы с ImGui.
- const GLuint & **getGLTexture** ()

  Получить текстуру для работы с OpenGL.
- void **updateTexture** ()

  Синхронизировать текстуру с текущим состоянием поля.
- void **unbindTexture** ()

  Закончить синхронизацию с полем.

4.2.3 Класс cellworld::FileSystem

Класс хранения файлов.

Состоит из 2 частей:

— Папка с файлами, содержащие информацию.

— Текстовый файл, хранящий имена файлов с информацией.

Открытые члены

— **FileSystem** (std::string txt_name, std::string folder_name)

Создаёт экземпляр, с файлом, хранящим имена, с именем txt_name+¨.txt¨ и папкой, хранящей файлы с именем folder_name.

— **FileSystem** ()

Создаёт экземпляр, с файлом, хранящим имена, с именем ¨file_names.txt¨.

— void **addFileName** (const char ∗file_name)

Добавить в экземпляр, файл с информацией с именем file_name.

— void **removeFileName** (int index)

Удалить имя файла из массива имён с данным индексом, и удалить сам файл из операционной системы.

— void **saveFileNames** ()

Сохранить массив имён в текстовый файл.

— void **loadFileNames** ()

Загрузить массив имён из текстового файла.

— void **checkFileNames** ()

Проверить валидность каждого из имён файлов.

— bool **findFileName** (const char ∗file_name)

Проверить наличие данного имени в массиве.

— std::string **getValidFileName** (std::string file_name)

Получить неиспользуемое имя, схожее с данным.

— const std::vector< std::string > & **getFileNames** ()

Доступ к чтению массива имён.

— const std::string & **getFolderName** ()

Доступ к чтение имени папки с файлами

Закрытые члены

— void **checkFileName** (int index)

Проверить валидность имени с данным индексом.

Закрытые данные

— std::string **store_names_file_**

Имя текстового файла, хранящий имена файлов с информацией.

— std::string **folder_name_**

Имя папки,хранящяя миры

— std::vector< std::string > **files_**

Массив имён.

4.2.4 Класс cellworld::Scenario

Класс сценария - расширенный класс поля

Открытые члены

- **Scenario** (int size_x=0, int size_y=0)
- void **makeNew** ()

  Очистить поле
- void **makeOneStep** ()

  Сделать 1 шаг симуляции
- void **spawnCreatures** (int amount)

  Создать данное количество существ на случайных позициях поля, использовать только на пустом поле. Также задаёт начальную популяцию.
- void **newCycle** ()

  Сделать новый цикл
- void **makeRewards** (Position, Position, float strength)

  Задать прямоугольник двумя позициями, за нахождение на котором существо каждый ход получает энергию=strength.
- void **giveRewards** ()

  Раздать награды
- void **CancelRewardsChange** ()

  Вернуться к предыдущему состоянию системы наград.
- void **resetRewards** ()

  Удалить все награды.
- void **rewardsEditor** (ImVec2 window_pos, ImVec2 window_size, float strength, ImTextureID texture)

  Ключевая часть графического интерфейса настройки наград, позволяет на пустом поле задавать прямоугольники, на которых будут награды.
- void **updateRewardsTexture** ()

  Синхронизирует графическое представление наград с текущим состоянием наград

Геттеры

- int **getInitialPopulation** ()
- long long **getIteration** ()

Открытые атрибуты

- int **cycle_len_**

  Длина цикла, находится в public, поскольку нужен доступ как для чтения так и для записи.

Друзья

- void **saveWorld** (const char *path, **Scenario** *current_field, const unsigned int &seed)

Функция сохранение мира

— void **loadWorld** (const char ∗path, **Scenario** ∗current_field, unsigned int &seed)

Функция загрузки мира

4.2.5 Класс cellworld::UI

Класс графического интерфейса. Является синглтоном. Содержит набор функций, которые представляют собой отдельные сцены. Сцены почти самостоятельны внутри себя, должны представлять возможность сменить сцену.

Открытые члены

— void **operator=** (const **UI** &)=delete
— **UI** (**UI** &)=delete

— void **updateWindowSize** (int width, int height)

Синхронизировать с размером окна

— void **loadScene** ()

Загрузить текущую сцену

Открытые статические члены

— static **UI** & **GetInstance** ()

Доступ к единственному экземпляру

Закрытые члены

— void **sceneUpdate** (int scene)

Сменить сцену на данную

— void **startScreen** ()

Функция для сцены стартового экрана

— void **CreationOfTheWorld** ()

Функция для сцены настройки/создания мира

— void **loadWorldScreen** ()

Функция для сцены выбора мира для загрузки

— void **SimulationOfTheWorld** ()

Функция для сцены симуляция мира

5 Техническая реализация

Описание того как работает программа.

Довольно большая часть реализации описана в документации, например, чтобы ознакомиться с классом хранения файлов см.(подраздел 4.2.3).

5.1 Структура

За исключение внешних библиотек, проект состоит из 4 локальных библиотек и исполняемого файла. Внутри всех локальных библиотек используется namespace::cellworld. Локальные библиотеки:

— creature - классы Creature и Field;
— scenario - класс Scenario + функции сохранения и загрузки мира;
— save_system - класс FileSystem;
— UI - класс UI;

Все библиотеки присоединяется к библиотеке UI, который уже присоединяется к исполнительному файлу.

5.2 Графический интерфейс

Для реализации используется фреймворк ImGUI с бэкендом GLFW + GLAD + OpenGL3. Класс UI хранит размер окна, номер текущей сцены, номер предыдущей сцены, общие данные для всех сцен, остальную работу выполняют сами сцены.

5.3 Как устроен класс Creature

Все существа имеют состояние. Возможные состояния:

— не существует(not_exist);
— мёртвое(dead);
— живое(alive).

Состояние	Не существует	Мёртвое	Живое
Позиция	Есть	Есть	Есть
Направление скорости	Нет	Есть	Есть, может менять
Модуль скорости	Нет	Есть	Есть, может менять
Энергия	Нет	Есть	Есть, может распоряжаться
Макс энергия	Нет	Нет	Есть
Цвет	Общий	Зависит от энергии	Зависит от генома
Масса	Нет	Нет	Зависит от генома
Нейронная сеть	Нет	Нет	Зависит от генома
Размножение	Нет	Нет	Есть

Таблица 5.1 — Различие между существами с разными состояниями

От чего зависит цвет существа, на что влияет энергия и макс энергия описано в подразделе 2.1.3.

Позиция

пара целых чисел x и y, за их валидностью следит поле.

Масса

Чем больше масса, тем больше макс энергия существа, импульс, пассивных расход энергии.

Направление и модуль скорости

Модуль определяет величину изменения позиции. Направление определяет какая координата меняется и в какую сторону.

Два существа оказались на одной позиции

Если оба существа живые, то умирает то существо, у которого меньше импульс(произведение массы на модуль скорости). Если одно существо мертвое, то другое существо занимает данную позицию и прибавляет к себе энергию первого существа.

Размножение

Если существо хочет размножиться, то сначала поле ищет соседнюю позицию на которой нет живого существа, и если она находится, то:

1) состояние существа на подходящей позиции меняется на живое;
2) родитель меняет геном на существа на свой с мутациями;
3) из генома существа берётся вес и родитель преобразует часть своей энергии в требуемый вес;
4) родитель дает половину от оставшейся энергии существу.

Для генерации мутаций генома используется детерминированный генератор случайных чисел с начальным сидом, заданный при создании мира.

Смерть существа

1) Состояние меняется на мёртвое;
2) Масса преобразуется в энергию;
3) Цвет меняется в зависимости от текущей энергии;
4) Модуль и направление скорости сохраняются.

Нейронная сеть

У всех живых существ нейросеть имеет вид:

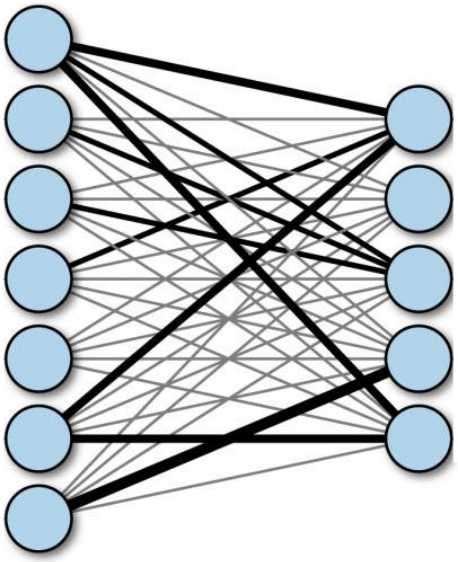

Рисунок 5.1 — Структура нейросети существа

Описание того какие нейроны за что отвечают в нейросети описано в подразделе 4.1.1. Выходные нейроны существа получаются с помощью перемножения матрицы входных нейронов и матрицы весов рёбер с помощью библиотеки Eigen3.

5.4 Как устроен класс Field

Задачи поля:

- Хранить состояние симулции;
- контролировать поведение, состояние, взаимодействие существ;
- Визуализировать симуляцию;
- UI - класс UI;

Поле имеет размеры. Позиции всех существ должны быть внутри поля,т.е. 0<=x существа <=ширина поля 0<=y существа <=высота поля Если это условие не выполняется, то x=x

Хранения состояние мира

За хранение состояния мира отвечают члены:

```
std::vector<Creature*> zoo_ptr_;
std::vector<Creature*> empty_zoo_ptr_;
std::vector<Creature> storage_;
```

storage_ - Хранилище существ, имеет постоянный размер = 2 * ширина поля * высота поля. zoo_ptr_ - набор указателей на текущее состояние, имеет постоянный размер = ширина поля * высота поля, zoo_ptr_[0] указывает на существо располагающееся на позиции (0,0) и так далее. empty_zoo_ptr_ - набор указателей на буферное состояние, имеет постоянный размер = ширина поля * высота поля, в начале симуляции заполнен указателями на существ с состоянием not_exist. Вся

работа с полем происходит через указатели, поскольку менять указатели местами(8 байт) намного быстрее, чем самих существ(>300 байт).

Визуализация

Для визуализации используется массив.

```
std::vector<unsigned int> colors_;
```

Хранит цвета всех существ текущего состояния, с помощью них OpenGL3 создаёт текстуру, которую уже выводит ImGUI.

5.5 Как устроен класс Scenario

Класс сценария - расширенный класс поля.

В расширенных функционал входят:

— создание живых существ со старта;
— сохранение и загрузка миров;
— циклический мир;
— награды -перед началом симуляции можно задать определённые позиции за расположение на которых, живое существо получает определённое количество энергии каждый ход;

создание живых существ со старта

void spawnCreatures (int amount) - создаёт на поле amount существ. Если amount> размер поля * ширина поля, то amount=размер поля * ширина поля. Создаёт перетасовку чисел от 0 до (размер поля * ширина поля-1). Из которой берётся первые amount чисел, из которых создаются позиции. На этих позициях создаётся живое существо со случайным геномом. initial_population_(Начальная популяция) = amount.

Функции сохранения и загрузки мира

Все данные поля, кроме существ и наград, сохраняются с помощью форматированного ввода. Сами существа и награды, которые могут составлять >99% значащей информации записываюся с помощью неформатированного ввода, что намного сильно улучшает ёмкость файла и скорость чтения и записи. Аналогично, все данные поля, кроме существ, загружаются с помощью форматированного вывода, а существа читаются с помощью неформатированного вывода. Запись существ в файл:

```
char* converted = new char[creature_size];
for (int i = 0; i < x * y; ++i) {
    Creature& current = current_field->getCreature(i);
    std::memcpy(converted, &current, creature_size - sizeof(NeuronNetwork));
    std::memcpy(converted + creature_size - sizeof(NeuronNetwork),
    current.networkPtr(), sizeof(NeuronNetwork));
    safe_file.write(converted, creature_size);
}
delete[] converted;
```

Циклический мир

Экземпляр класс Scenario следит, прошло ли достаточное количество итераций, чтобы перезагружать мир, если да, то выполняется функция void newCycle().
Функция схожа с функцией spawnCreatures, однако в ней только 1/8 существ от начальной популяции являются существами со случайным геном. Остальные 7/8 существ становятся детьми выживших в коцне цикла существ, причём программа стремится равномерно распределить количество детей для каждого выжившего. Если часть детей не получается равномерно распределить(кол-во оставшихся детей< кол-во выживших), то случайно выбранная группа выживших получает ещё 1 ребёнка.

Награды

Разделена на 2 части:

- логическая часть - массив, хранящий сколько энергии сущетсво должно получить энергии за нахождении на данной позиции);
- графическая часть - является текстурой, репрезентирующая массив, в которой каждое число массива соответствует определённому цвету, идентична цветам энергии мёртвых существа(см. рис. 2.5).

Всё работа происходит в логической части, а графическая часть просто синхронизируется с логической. Функции связанные с наградами упомянуты в документации класса в подразделе 4.2.4.

5.6 Что происходит за 1 шаг симуляции

Функция одного шага симуляции

```
void Scenario::makeOneStep(){
    updatePositions();
    updateStates();
    giveRewards();
    ++iteration_;
    if (cycle_len_>0 && iteration_ % cycle_len_==0){
        newCycle();
    }
}
```

Все функции, внутри тела уже были описаны ранее.