ТЕХНОЛОГИЧНО УЧИЛИЩЕ ЕЛЕКТРОННИ СИСТЕМИ

към ТЕХНИЧЕСКИ УНИВЕРСИТЕТ - СОФИЯ

Д И П Л О М Н А Р А Б О Т А

Тема: Стратегическа игра разработена с Unity3D

Дипломант:

Владимир Младенов

Научен ръководител:

Виктор Кетипов

С О Ф И Я

2 0 1 8

Увод

Историята на видео игрите датира още от 1950 година, когато Джоусеф Кейтс създава първата форма на морски шах[1]. Популярността на видео игрите, обаче, става факт едва през седемдесетте и осемдесетте години на миналия век, когато на публиката биват представени аркадните игри, конзолите и джойстиците, както и графики на компютърните екрани[2]. От тогава, популярността на игрите расте постоянно. Този растеж, им позволява да стават все повече и по- развити. Постепенно, игрите започват да се докосват до умения от живота извън виртуалното пространство, а наред с това, непряко развиват редица умения на играчите. Действие в правилния момент, организация, интериорен дизайн, управление на ресурсите и основен морал, са само някои от уменията, които игрите развиват в играчите[3].

В тази връзка, **целта** на дипломната работа е да се създаде стратегическа игра, в която играчите трябва да използват познания по логистика и ефикасно управление на ресурсите. Играта трябва да позволява на играча да:

- Движи своя герой из един процедурно генериран свят и да му позволява да добива ресурси и да създава устройства за добивът и обработката на тези ресурси.
- Свързва вече създадените устройства, за да създаде по- сложни механизми и продуктови линии и така да постигне пълна автоматизация на производството.

[1] Google Newspapers- Ottawa Citizen (October 8th, 1975)
[2] Wikipedia- History of video games
[3] Cracked.com- 5 Real Skills Video Games Have Secretly Been Teaching Us

- Позволява на играчът да отключва нови устройства и по този начин да напредва с нови технологии и артикули.
- Позволява на играчът да съхранява своите артикули в инвентар.

Успешното реализиране на поставената цел е свързано с изпълнението на следните основни задачи:

- След проучване на подходящ алгоритъм, същият да се използва за имплементиране на процедурно генериран свят и ресурси.
- Успешно да бъде имплементиран инвентар и да бъде създаден подходящ потребителски интерфейс, който да позволява на играча да контролира своите артикули.
- Да бъдат създадени модели на устройствата и да се създаде подходящ потребителски интерфейс за всяко от тях.
- Да бъде създадена система за отключване на нови устройства, и да бъде създаден подходящ потребителски интерфейс, който играча да може да използва.

Глава 1. Преглед на съществуващи игри и известни развойни среди.

1.1 Преглед на идентични стратегически игри

1.1.1 Factorio

Factorio[4] е игра в която строиш и поддържаш фабрики. В играта може да се добиват ресурси, да се проучват нови технологии, да се строи инфраструктура, да се автоматизира производството и да се води битка с врагове. Идеята е да се използват основни познания по логистика, за да се комбинират основни артикули и машини в огромни производствени линии. Играта е много стабилна и добре оптимизирана да поддържа огромни фабрики. Основния език на програмиране е Lua, а това позволява и лесно писане на модове от потребителите. Има поддръжка за игра от няколко играча на един сървър. В играта са имплементирани няколко различни режими на игра:

- Кампания- този режим съдържа няколко групи от мисии, които играча да изпълни. Основните са два:
  - First steps[5]- Играча попада на неизвестна планета. Трябва да се научи да създава първите си инструменти и машини, докато се предпазва от враговете, които го обкръжават (Фигура 1-1).

[4] Factorio Web Site
[5] Factorio Wikipedia- First Steps campaign

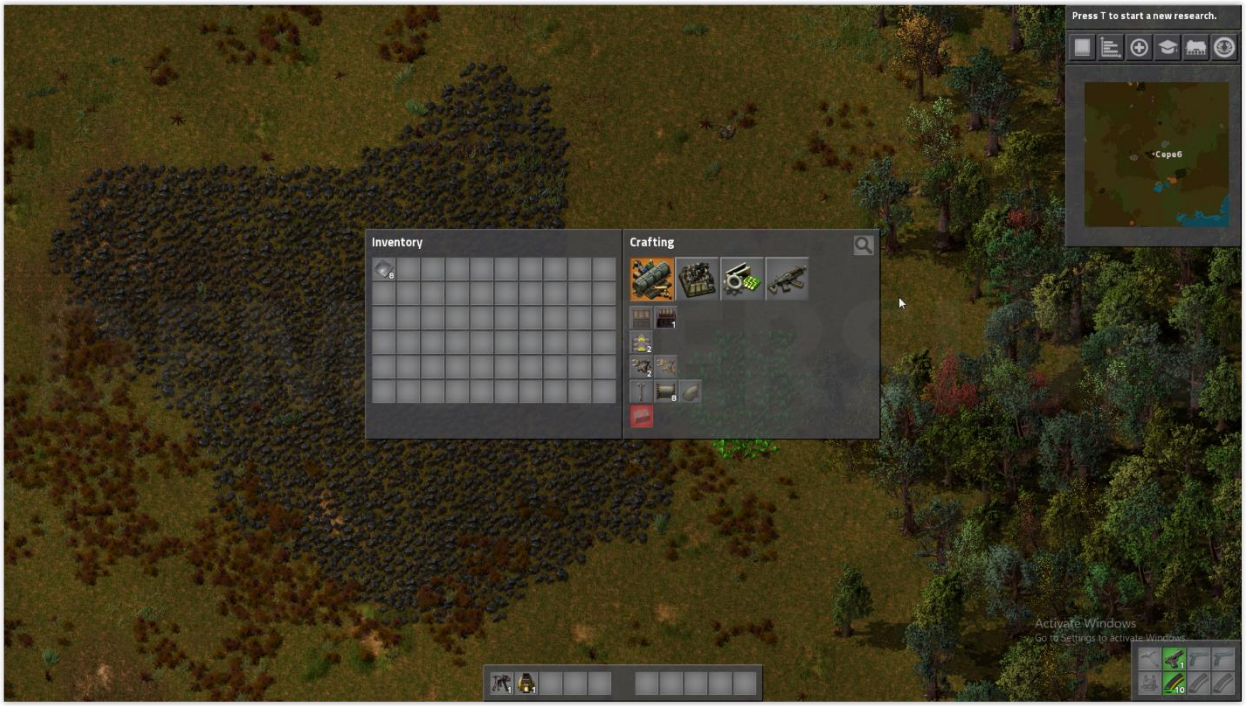
Фигура 1-1. Играчът на непозната планета, близо до останките на кораба му.

- New hope[6]- Продължение на First steps. След като играча е открил останките на паднал кораб, идва трудната част. За да започне сериозна колонизация на планетата, трябва да се вземе информацията от кораба. Тя ще позволи проучването на по- сложни технологии и ще даде информация за обграждащата зона. Лошата новина е, че има огромна вражеска база около кораба.

- Свободен режим- Играчът започва с осем железни плоскости, машина за добив на ресурси, и печка. Играча разполага с пълна свобода и безкраен свят да развива своите фабрики.

[6] Factorio Wikipedia- New Hope campaign

Играчът разползага с инвентар, който му позволява да съхранява артикули, както и да създава нови артикули и машини (Фигура 1-2).



Фигура 1-2. Играча разполага с инвентар.

Играта предоставя механизми за свързване на машините, така че да има автономност при производството (Пример: Машината за добив на ресурсите се свързва с печката, която се свързва с механична ръка, слагаща вече обработения ресурс в ковчеже (Фигура 1-3). По този начин системата може да работи и без играча да е наблизо).

Фигура 1-3. Автоматизирана система за добив на желязни плоскости.

Крайната цел на играта е да се построи ракетна база и да се изтреля ракета. Въпреки, че целта на играта е изпълнена, играча може да продължи да играе и след това.

1.1.2 Infinifactory

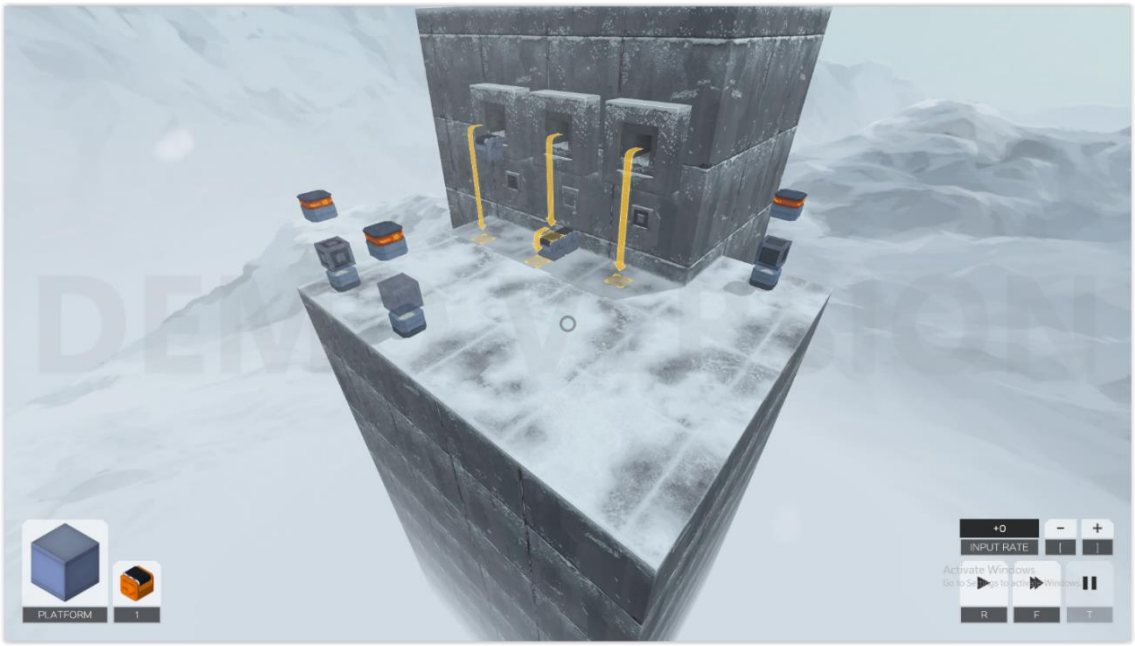
Infninfactory е пъзел игра, в която играча има пълна свобода да строи фабрики, произвеждащи продукти за извънземните, които са го пленили, опитвайки се да оцелее. Целите на играта са няколко:

- Да се измислят и поддържат фабрики в един напълно триизмерен свят.
- Да се оптимизират решенията и след това да се оптимизират още, за да може играча да се класира на 1- во място сред приятелите си.
- Да премине кампания с 50+ пъзела.
- Дори и след кампанията, играча може да създава собствени нива и решения.

Играта е създадена от Zachtronics, създателите на SpaceChem и Infiniminer[7].

При започване на решението на даден проблем, т.е. при стартиране на ново ниво, играча се запознава с входа и изхода на линията за производство, като неговата задача, е да ги свърже, така че да покрие дадено изискване. Първото ниво на играта представлява линия с три различни входа- три различни блокчета, които трябва да се свържат с три различни изхода (Фигура 1-4).

[7] Zachtronics.com- Infinifactory

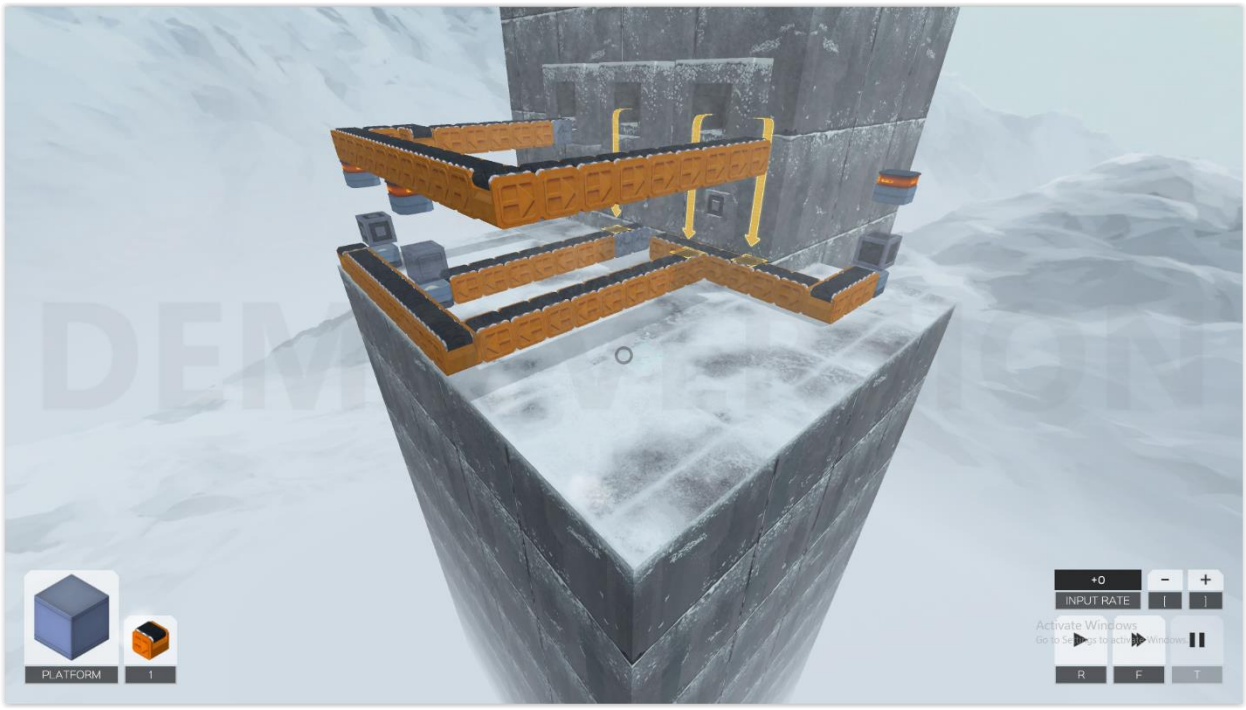


Фигура 1-5. Начало на първо ниво.

Всеки изход, има определено изискване. В този случай, всички изходи, искат за всяко, общо десет блокчета от желаните да бъдат доставени (Фигура 1-5).

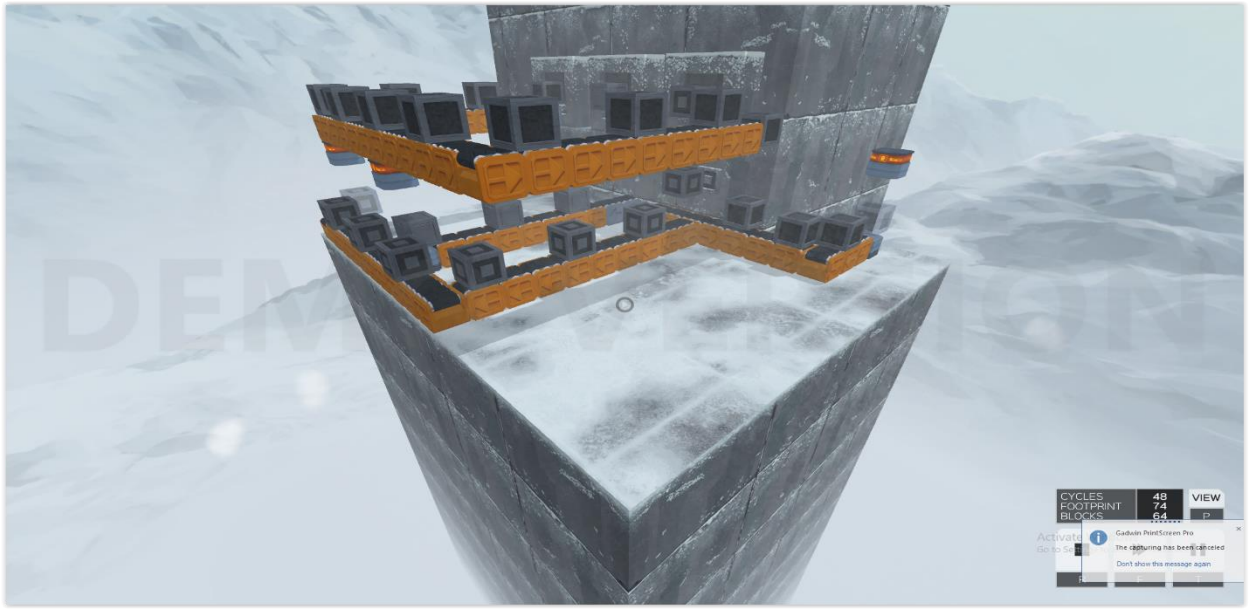
Фигура 1-4. Изход- изискването е 10 блокчета да бъдат доставени.

След запознаване с изискванията, играчът построява продуктовата линия (Фигура 1-6).



Фигура 1-6. Построена продуктова линия.

След това играчът само трябва да я стартира и да достави желаното количество до изходите. Решението е валидно, ако това се случи (Фигура 1-7).



Фигура 1-7. Валидно решение на проблема.

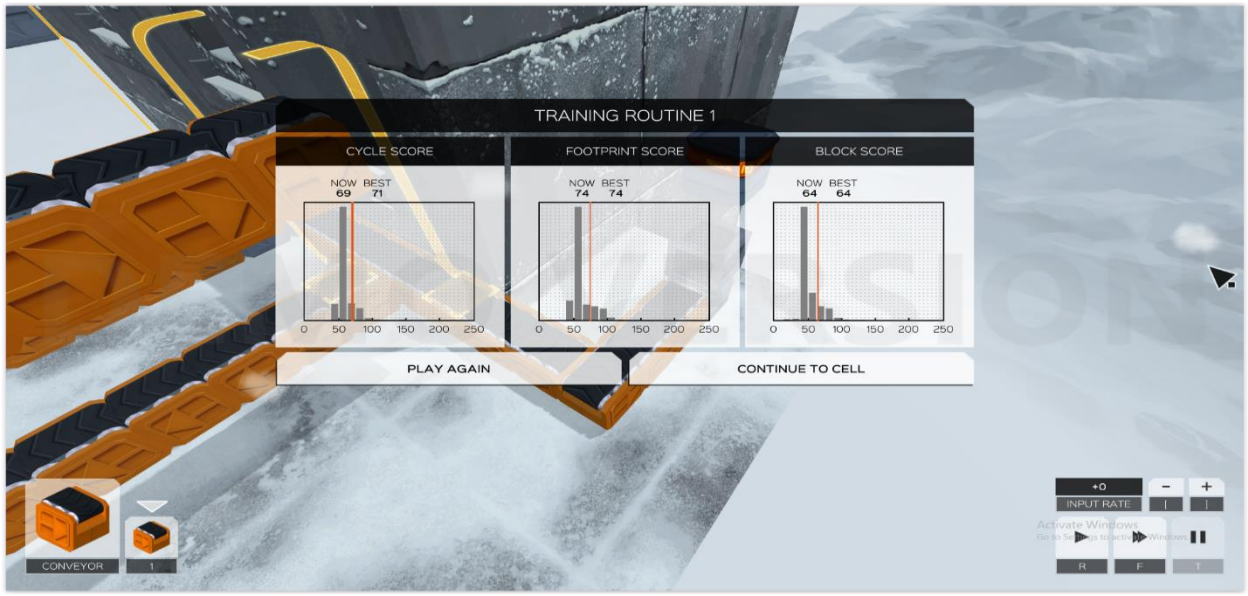
След като се представи решение, се показват графики с резултати на играчът и обозначените точки за всяка категория на решението, спрямо тези на други играчи (Фигура 1-8). Играчът след това може да представи други решения.

1.2 Преглед на известните развойни среди

Тъй, като обекта на дипломната работа се разработва, чрез гейм енджин, в тази точка ще бъдат разгледани известните гейм енджини.

1.2.1 Unity

Unity е една от най- популярните гейм енджини в момента. Тя е разработена от Unity Technologies. Притежава голяма поддръжка от платформи, за които може да се разработват игри- Windows, Android, VR и други. Предимно се използва за разработка на 2D и 3D игри, както и за симулации за компютри, конзоли и мобилни приложения.



Фигура 1-8. Графиките след представяне на решение.

Unity поддържа 2D и 3D графики и използване на скриптове, написани на езика C#. Още два езика са били поддържани- Boo, който е бил премахнат с версия 5 на Unity и JavaScript, чийто процес на премахване е започнал през август 2017 с излизането на Unity 2017.1

Енджинът използва следните графични библиотеки:

- Direct3D за Windows и Xbox One
- OpenGL за Linux, macOS и Windows
- OpenGL ES за Android и Ios
- WebGL за уеб приложения
- И удобни библиотеки за конзолите

За 2D игрите, Unity позволява да се качват 2D изображения, както и подобряване на 2D текстури. За 3D, Unity предлага различни опции за компресиране на текстурите, избор от различни визуализации, както и възможност за промяна на резолюцията за различните платформи.

Като лицензи, Unity предлага четири вида:

- Персонален- Това е стандартният лиценз. Той предлага стандартните Unity услуги, т.е. всички възможности на енджина и всички платформи, но има граничен приход от $100,000. Освен това, не се предлагат други Unity услуги, като получаване на доклад за представянето на играта, премиум подкрепа, както и достъп до сорс кода. Този лиценз е безплатен.
- Плюс- Предлагайки по- висока приходна граница от $200,000 и повече Unity услуги, този лиценз е платен за $35 на месец. В него е добавена услуга за доклад за представянето на играта.
- Про- Без приходна граница, и добавена премиум подкрепа, този лиценз може да бъде закупен за $125 на месец.

- Ентърпрайз- Най- пълният лиценз на Unity- предлага всички услуги. Цената е по договаряне[8].

1.2.2 Unreal Engine

Unreal Engine е разработен от Epic Games. Първата поява на този енджин е през 1998, с играта Unreal. Въпреки, че предимно се използва за разработка на игри със стрелба от първо лице, успешно е използвана и за редица други жанрове (MMORPG, RPG и др.)

Тъй като използва C++ като език на разработка, енджинът предоставя висока портативност и това я прави развойна среда, която много разработчици използват днес. Всъщност, C++ не е първичният език на програмиране за Unreal Engine. Преди Unreal Enigne 4, се е използвал езикът UnrealScript, познат още като UScript.

Платформите, които енджинът поддържа са Windows, macOS, Linux, SteamOS, HTML5, iOS, Android, Други.

Unreal Engine печели много награди, а през 2014, печели Гинес рекорд за най- успешен гейм енджин[9].

1.2.3 CryEngine

CryEngine, е разработен от Crytek. Първоначалната версия на енджинът е използвана за разработката на Far Cry и продължава да се подобрява, за да поддържа нови конзоли и хардуер за игрите.

[8] Wikipedia- Unity game engine
[9] Wikipedia- Unreal engine

Warehouse Studios използва модифицирана версия на енджинът за Kingdom Come: Deliverance, а Ubisoft поддържа доста модифицирана версия на CryEngine- Dunia Engine, който те използват за следващите версии на FarCry.

Езиците, които енджинът поддържа, като налични за разработка са C#, C++ и Lua[10].

Известни заглавия разработени, чрез CryEngine са:

- Crysis поредицата
- FarCry
- Homefront: The Revolution
- И други

[10] Wikipedia- CryEngine

Глава 2. Изисквания към продукта технологии за разработка.

2.1 Дефиниране на изискванията към продукта.

Следните неща следва да бъдат имплементирани:

- Възможност за контрол на играча и движение във всички посоки на процедурно генерирания терен.
- Възможност за добив на ресурси.
- Система от артикули, които играчът да може да притежава.
- Инвентар, както и основна функционалност на артикулите в инвентара (местене, разделяне, премахване от инвентара). Достъп на играча до инвентара му.
- Възможност за създаване на машини за добив и управление на ресурсите и взаимодействие на играча с машините, чрез натискане върху машината.

2.2 Избор на развойна среда

За развойна среда е избран Unity3D.

Основанията за избор тази развойна среда са следните:

- Unity като цяло предоставя по- благоприятна среда за разработка, която е подходяща за неопитните в областта на програмирането на игри- има много онлайн материали, чрез които един разработчик-новак може да научи основите на гейм енджинът. Освен това, общността от разработчици, които използват Unity е голяма, и това позволява, разработчик да се запознае с многобройни решения на един проблем, както и да черпи идеи от многобройните игри и демота, които разработчиците публикуват в онлайн форумите.

- Голямо удобство представлява и онлайн магазинът за артикули, направени от други потребители. Могат да се използват безплатни материали, модели и други.
- В плюс на Unity, е и програмният език, който той използва- C#. Езикът е обектно ориентиран, а на база материала, изучен по време на учебните занятия, това се превръща в чудесен избор, като език за разработка.
- Освен трите причини, описани по- горе, реалният опит придобит от лятната производствена практика- а именно разработка на игри с Unity, поставя този енджин на първо място като избор.

Като софтуер за моделиране, текстуриране и анимиране е избран Blender, поради удобство и предишен опит.

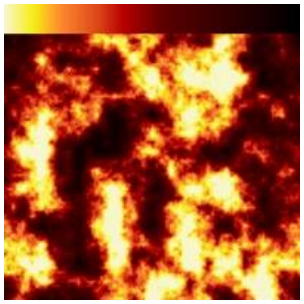
Като софтуер за създаване на 2D текстури е избран Photoshop, поради множеството функции, които предлага.

2.3 Избор на алгоритъм за генериране на терена

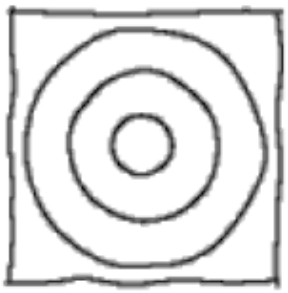
За създаване на терена, се използва Перлинов шум. Този алгоритъм предоставя безкрайна поредица от псевдо- случайна информация. Чрез него се създава процедурно генерирано съдържание- процедурно генериран терен (Фигура 2-1[11]), различни ефекти (Фигура 2-2[11]), илюзия за ръчно написани линии (Фигура 2-3[11]) и други [11].

[11] Flafla2 @ Github.io- Understanding Perlin Noise

Фигура 2-1. Процедурно генериран терен

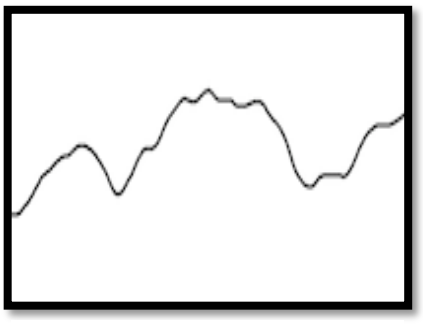


Фигура 2-2. Огнен ефект



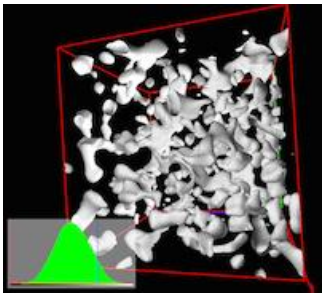
Фигура 2-3. Илюзия за ръчнонаписани линии

Перлиновият шум е градиентен- той е изграден от градиентни вектори, които комбинирани заедно образуват стойностите на шума във всяка негова точка. Най- често се използва за едно (Фигура 2-4[12]), две (Фигура 2-5[12]) и три (Фигура 2-6[12]) измерения, като чрез различните измерения могат да се постигнат различни ефекти [12].



Фигура 2-4. Перлинов шум- едно измерение

Фигура 2-5. Перлинов шум- две измерения

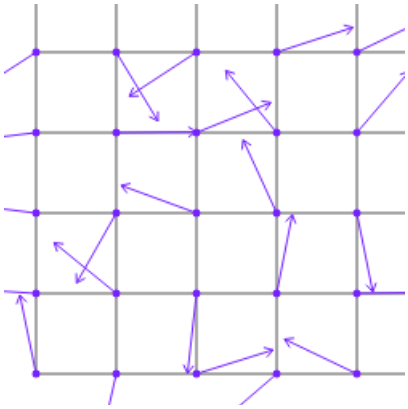


Фигура 2-6. Перлинов шум- три измерения

[12] Flafla2 @ Github.io- Understanding Perlin Noise

За създаване на Перлинов шум се изпълняват няколко стъпки:

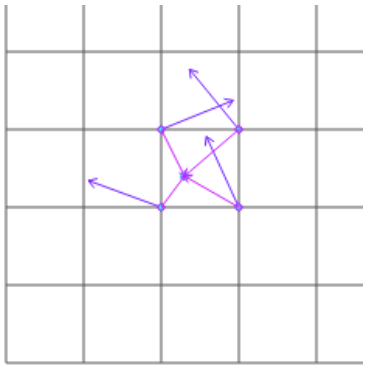
1. Представя се мрежова координатна система, при която по X и Y осите, се налагат цели числа. За всяка точка от тази система, се създава градиентен вектор, който сочи в случайна посока (Фигура 2-7[13]). Векторите са градиентни, защото шумовата функция се увеличава с посоката на вектора[13].



Фигура 2-7. Координатна система, изпълнена с градиентни вектори.

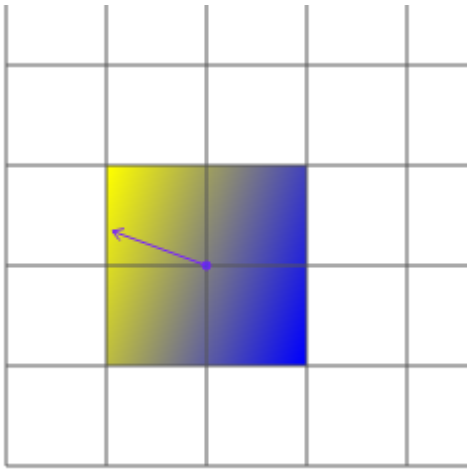
2. За всяка точка от шума, която се търси, се изчислява функция на базата на четирите обграждащи точки- Q (Фигура 2-8[13]) и четирите градиентни вектора- G. За всяка Q от четирите точки, се изчислява скаларното произведение на градиентният вектор на Q и векторът, отбелязващ разстоянието от точката до Q[13].

[13] Huttar.net- Learning how Perlin noise works



Фигура 2-8. Точка от координатната система и векторите от точките Q

Полученият градиент от скаларното произведение е представен на Фигура 2-9[14].



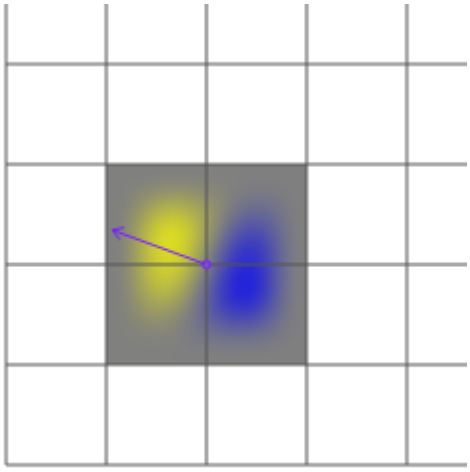
Фигура 2-9. Полученият градиент чрез скаларното произведение.

[14] Huttar.net- Learning how Perlin noise works

3. Финалната стъпка на алгоритъма, е да се приложи преливащ филтър, за да може резултата от комбинираните градиенти от всяка точка, да изглеждат естествено. Този филтър се изчислява, чрез следната функция:

$$6t5 - 15t4 + 10t3$$

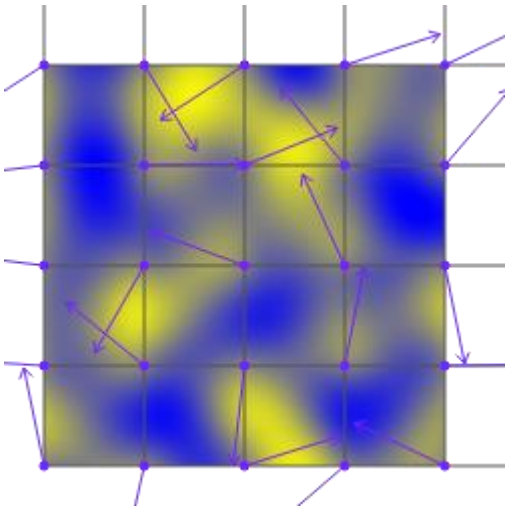
Тази функция се прилага на локалните за квадрата, образуван от четирите обкръжаващи точки на търсената точка, координати (т.е. ако търсената точка е с координати (124.4, 23.2), локалните координати са (0.4, 0.2)). Полученият градиент, след прилагане на филтъра е представен на Фигура 2-10[15].



Фигура 2-10. Филтрираният градиент.

[15] Huttar.net- Learning how Perlin noise works

След изпълнение на тези три стъпки за всяка точка от координатната система, се получава цялостна картина на шума (Фигура 2-11)[16].



Фигура 2-11. Резултатният шум.

В зависимост от целта на използването на Перлиновия шум, могат да бъдат променени честотата, амплитудата, както и октавите на шума.

Лесното интегриране и използване на този алгоритъм, е причина за изборът му като алгоритъм за процедурно генериране на терена на играта. Възможостта за манипулация чрез промяна на честотата, амплитудата и октавите, е подходящо за терен. Тези промелниви могат да се променят лесно и бързо и по- този начин терена да се модифицира по желан начин.

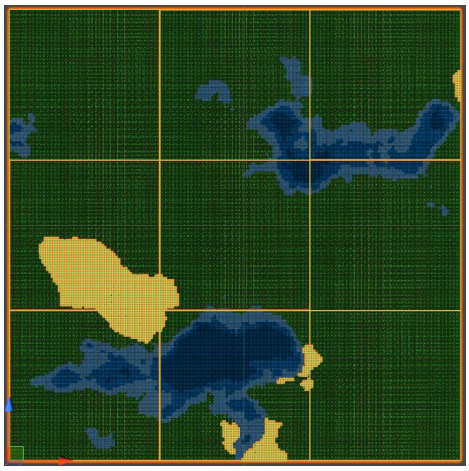
[16] Huttar.net- Learning how Perlin noise works

Глава 3. Реализация на стратегическа игра на Unity3D

В тази глава е описана реализацията на обекта на дипломната работа. На лице са процедурно генериран терен, чрез модифициране на Перлинов шум, лесно достъпен и контролиран инвентар с размер 100 слота, хотбар на играча за бърз достъп до важни артикули, устройства за добив и обработка на ресурсите, система за създаване на машини, както и система за отключване на нови устройства.

3.1 Процедурно генериран терен

Терена на играта се състои от частици, които наричаме парчета (Фигура 3-1, Фигура 3-2). Тези парчета са съставени от 4,096 плочки, разделени на 64x64 по X и Z оста на света. Всяка плочка е с размер 10x10 юнити единици.



Фигура 3-1. 3X3 парчета.

Фигура 3-2. Едно парче.

3.1.1 Процедурно генериране на терена

Тъй като, генерирането на безкраен терен, е физически невъзможно, процедурно се генерират парчета, в регион с определен радиус около играча.

Така се постигат две неща:

- Оперативната памет не се претоварва, защото се запълва бавно и постепенно с минимално количество данни. По този начин значително се подобрява оптимизацията на играта.
- Процедурното генериране на терена не влияе по никакъв начин на действието на играча, защото винаги има генерирани парчета около него.

За процедурното генериране на терена се грижи скриптът ChunksController.cs. Този скрипт е закачен за празен обект в играта. Идеята му е да генерира парчета около играча, когато той се движи. В началото на играта автоматично се генерират определен брой парчета около играча, а когато играчът за пръв път стъпи на дадено парче, се извиква функцията GenerateChunksAround(Фунцкия 3-1), която генерира парчета около него.

```
//Generate the chunks around the player
    public void GenerateChunksAround(Vector2 coordinates)
    {
        List<Vector2> notGeneratedChunks = GetListOfNotGeneratedChunksAround(
                                                                coordinates);
        for (int i = 0; i < notGeneratedChunks.Count; i++)
        {
            InstantiateChunkAt(notGeneratedChunks[i]);
        }
    }
```

Фунцкия 3-1. GenerateChunksAround()

Това става, като за всяко негенерирано място около играча, се извиква функцията InstatiateChunkAt(Фунцкия 3-2), която създава парче на дадените координати, които са в опростен вид. След като парчето бъде създадено, се извиква функцията InitializeChunk на прикачения за него скрипт Chunk.cs.

```
//Generate the chunk and initialize its mesh
    private void InstantiateChunkAt(Vector2 coordinates)
    {
        GameObject instance = Instantiate(_chunkPrefab);
        instance.transform.position = new Vector3(
                                        coordinates.x * _chunkSize,
                                        0f,
                                        coordinates.y * _chunkSize);
        instance.name = "Chunk " + coordinates;
        instance.transform.parent = chunkWrapper.transform;
        instance.GetComponent<Chunk>().InitializeChunk(
                                        new Vector2(coordinates.x, coordinates.y));
        _generatedChunks.Add(instance);
    }
```

Функция 3-2. InstantiateChunkAt()

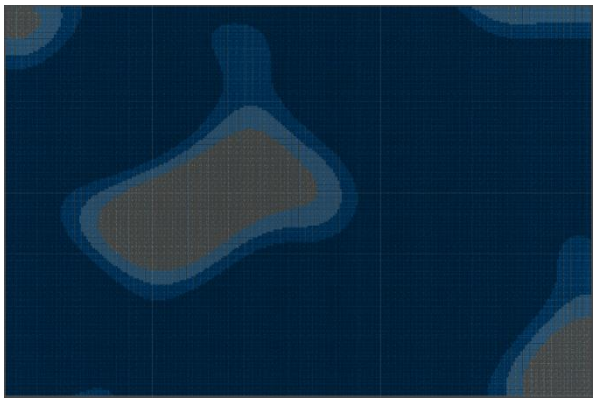
3.1.2 Генериране на плочки и текстури за парчетата

За генериране на плочките и текстурите на парчето, се грижи прикачения за обекта скрипт- Chunk.cs. Функцията InitializeChunk(), се грижи за инициализиране на стойността на променливата, която съдържа координатите на съответното парче и извиква метод за генериране на плочките и текстурите.

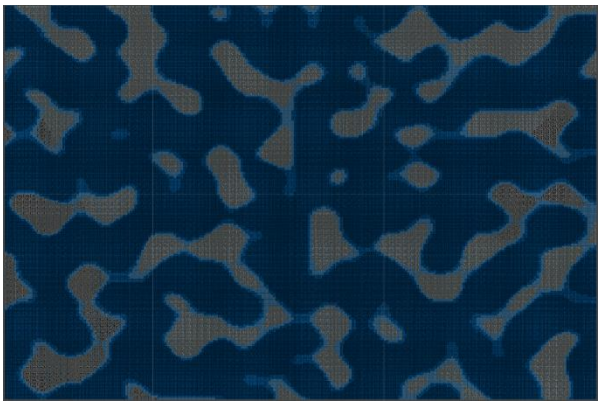
Първото, което се прави е да се инициализират плочките. За определяне на типа на всяка отделна плочка във всяко парче, се използва вградената в Unity функция Mathf.PerlinNoise(float x, float y), чиято стойност се връща от Перлиновия алгоритъм описан във втора глава (2.3), където X и Y са координатите на точката, чиято стойност търсим. Съответните координати се модифицират като се разделят на определена честота, по- този начин контролираме честотата на шума. Контрола върху честотата, определя колко често искаме да има зони от даден тип плочки– в случая водни. Получената формула е следната:

$$perlinValue = Mathf.PerlinNoise(x \,/\, frequency, y \,/\, frequency)$$

Самото контролиране на честотата, не ползволява контрол над големината на зоните. При по голяма честота се наблюдават много големи зони (Фигура 3-3), образуващи една голяма водна зона. Докато при по- малка честота, се получават малки зони, но по- чести(Фигура 3-4).



Фигура 3-3. Водни зони с честота 150.

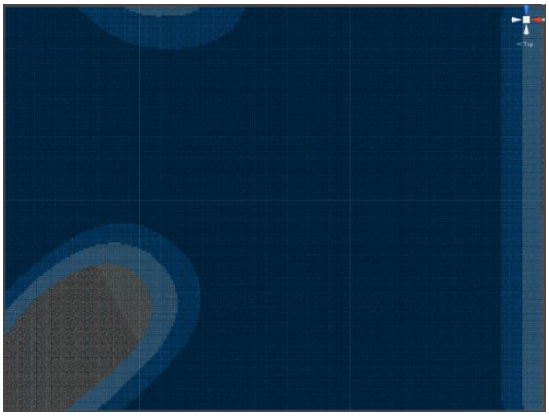


Фигура 3-4. Водни зони с честота 70.

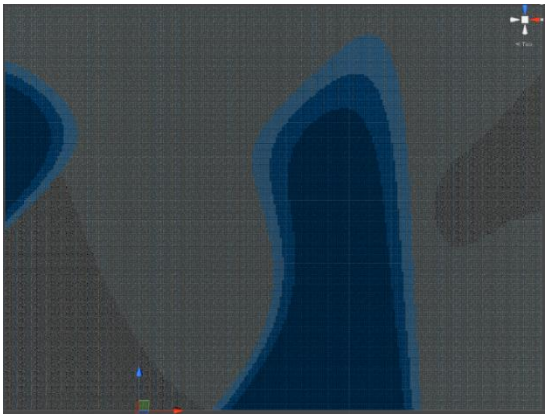
Проблемът с контролирането на големината на зоните се разрешава, като стойноста от шума за дадена точка се умножи по модификатор, който наричаме амплитуда. Така след като е променена честотата на зоните, можем да контролираме големината им, за да получим нужния резултат. Формулата се изменя във вида:

$$perlinValue = Mathf.PerlinNoise(x \,/\, frequency, y \,/\, frequency) * amplitude$$

При по- малка амплитуда се получават по- големи зони(Фигура 3-5), а при по- голяма– по- малки(Фигура 3-6).

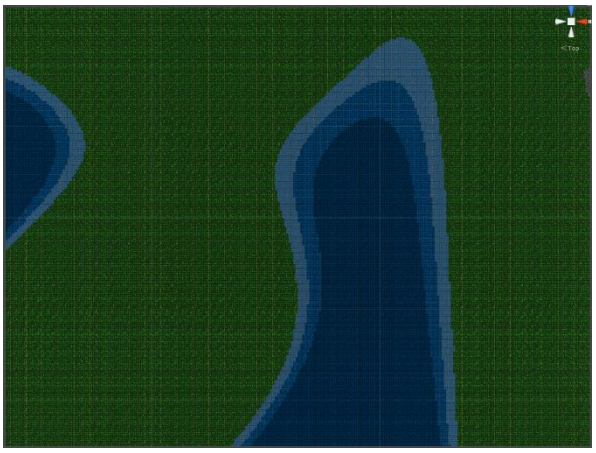


Фигура 3-5. Честота 150. Амплитуда 1.



Фигура 3-6. Честота 150. Амплитуда 2.

Друг проблем, който се забелязва е формата на тези зони. Наблюдават се закръглени форми, които нямат натурален вид(Фигура 3-7).



Фигура 3-7. Честота 150. Амплитуда 2. Октави 1.

Това се получава, защото Перлиновият шум е изграден от градиенти и преминаването от една стойност в друга става плавно. За да се разреши този проблем, се наслагват различни октави на шума, т.е. постепенно се наслагват няколко шума с различни честоти и амплитуди и по- този начин се получават по- натурални форми (Фигура 3-8).

Фигура 3-8. Честота 150. Амплитуда 0.75. Октави 6.

Финалният вид на функцията(Фунцкия 3-3) за намиране на стойност от перлиновия шум изглежда така:

```
private static float GetPerlinNoiseValueAt(float x, float y, float frequency, int
octaves, float amp)
    {
        float returnValue = 0f;

        float gain = 1.0f;
        for (int i = 0; i < octaves; i++)
        {
            returnValue +=
              Mathf.PerlinNoise(x * gain / frequency, y * gain / frequency)
              * amp / gain;
            gain *= 2.0f;
        }

        return returnValue;
    }
```

Фунцкия 3-3. GetPerlinNoiseValueAt()

Плочките могат да бъдат различни видове в зависимост от стойността, която алгоритъма връща за съответстващите координати. В програмата, се дефинира променлива от тип двумерен масив, която пази информация за всички плочки в парчето. При създаване на парче, масивът се инциализира (Фунцкия 3-4) и се генерира текстура на терена, в зависимост от стойностите му.

```
private void InitializeDataMap()
    {
        //Initialize different tile types maps
        TilesEnum[,] _biomesMap =
        PerlinNoiseGenerator.GenerateBiomesMap(_chunkCoordinates);
        TilesEnum[,] _waterMap =
        PerlinNoiseGenerator.GenerateWaterMap(_chunkCoordinates);
        bool[,] _ironOreMap = PerlinNoiseGenerator.GenerateIronOreMap(_chunkCoordinates);
        bool[,] _coalMap = PerlinNoiseGenerator.GenerateCoalMap(_chunkCoordinates);

        for (int x = 0; x < 64; x++)
        {
            for(int y = 0; y < 64; y++)
            {
                if (_waterMap[x, y] != 0) //If there is water on this tile
                {
                    _tilesDataMap[x, y] = new TileData(_waterMap[x, y]);
                    _notWalkableMap[x, y] = true;
                }
                else if (_ironOreMap[x, y]) //Else if there is iron on this tile
                {
                    _tilesDataMap[x, y] = new TileData(
                                                TilesEnum.IRON_ORE,
                                                NewResource(TilesEnum.IRON_ORE, x, y));
                }
                else if (_coalMap[x, y]) //Else if there is copper on this tile
                {
                    _tilesDataMap[x, y] = new TileData(
                                                TilesEnum.COAL,
                                                NewResource(TilesEnum.COAL, x, y));
                }
                else //Else get the biome map tile
                {
                    _tilesDataMap[x, y] = new TileData(_biomesMap[x, y]);
                }
            }
        }
    }
```

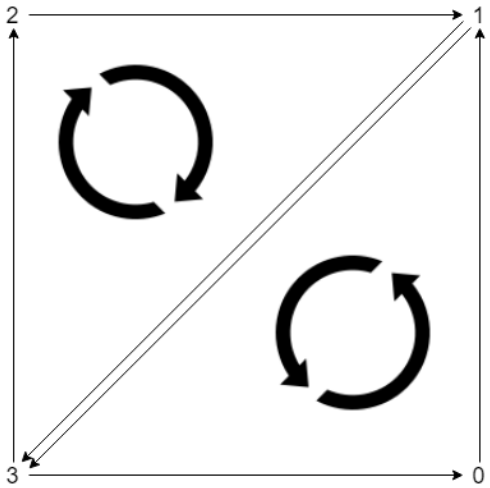
Функция 3-4. InitializeDataMap()

Вторият етап е за генерирането на текстурите спрямо вече инициализираните в първи етап плочки. За това се грижи функцията GenerateTexture.

Функцията итерира матрицата от плочки, и за всяка плочка изпълнява следната последователност от стъпки:

1. Превръща опростените координати на плочката в глобални, т.е. в оригиналните юнити координати.

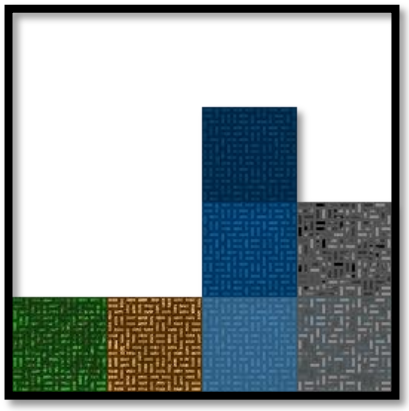
2. След което се създават четири на брой вертекси(в геометрията, вертекс е точка, където две или повече криви, прави или ръбове се срещат.[17]). Тези вертекси са върховете на плочката. Те служат за рамка на текстурата.
3. Вертексите обаче не са достатъчни за да се построи една текстура. Следващата стъпка е да се дефинира направление по което да се изрисува текстурата. Квадратът дефиниран от четирите вертекса се разделя на два триъгълника– полигони. В листа с цели числа triangles, се записват числа, които са индекс означаващ вертексите от точка 2, в последователността в която искаме направлението да се движи. В нашия случай всеки квадрат се строи по начина отбелязан на (Фигура 3-9). Като начало се отбелязват три точки- 0, 1, 3. Това построяване дефинира първия триъгълник. Следващата тройка числа- 1, 3, 2 дефинира другият триъгълник.



Фигура 3-9. Начин за построяване на един квадрат- плочка.

[17] Wikipedia- Vertex (geometry)

4. След като са дефинирани вертексите и триъгълниците, може да се нарисува текстурата върху плочката. За тази цел преди това трябва да съществува така наречения spritesheet, т.е. лист или таблица, съдържаща различните текстури за различните плочки(Фигура 3-10).



Фигура 3-10. Spritesheet на текстурите на терена.

Всички текстури на плочките са събрани на един лист, като се приема, че всяка текстура има индекс (например: тревната е с индекс 0, саванната с индекс 1 и т.н.). Когато се означават координатите на текстурата, е необходимо да се означат четирите върха на квадратчето към което спада текстурата. За да се случи това, трябва да се дефинира единица, която се равнява на страната на един квадрат. Тъй като в листа с координати, най- голямата възможна стойност е 1, а най- малката 0, следва, изчилсляването на тази единица да става, чрез следната формула:

$$unit = 1 \,/\, spriteSheetSize$$

Където spriteSheetSize е размера на листа, дефиниран от броя текстури по оста X или Y (листа е квадрат и двете стойности са равни). За момента в играта съществуват общо седем вида плочки, разпределени по листа, а неговия размер е 4. Чрез формулата се изчислява, че една единица е 0.25. След като е намерена единицата, трябва да се намерят координатите на текстурата, като за тази цел използваме и предефиниран идентификатор за текущата плочка, който получаваме при инициализацията на плочките. Първо намираме отношението на идентификатора към размера на листа.

$$idToSpriteSheetSize = id \ / \ spriteSheetSize$$

Пресметнатото отношение за текстура с индекс 6 е 1.50. X координата се получава като се вземе само десетичната част на отношението, т.е. след като отношението е 1.50, X координатата е 0.50. Y координатата се получава, като цялата част на отношението се умножи по вече намерената единица на листа (unit). Така 1.50 отношение означава, че Y координата е 0.25. Чрез тези координати откриваме исканата текстура в листа- текстурата с координати (0.50, 0.25), т.е. нормална вода.

Последният етап е да се запишат листовете за вертексите, триъгълниците и текстурите в MeshRenderer компонента (компонент, който се грижи за чертаенето на текстурата и формата на обектите) на парчето.

Тази последователност от стъпки се изпълнява от фунцкията GenerateTexture (Фунцкия 3-5).

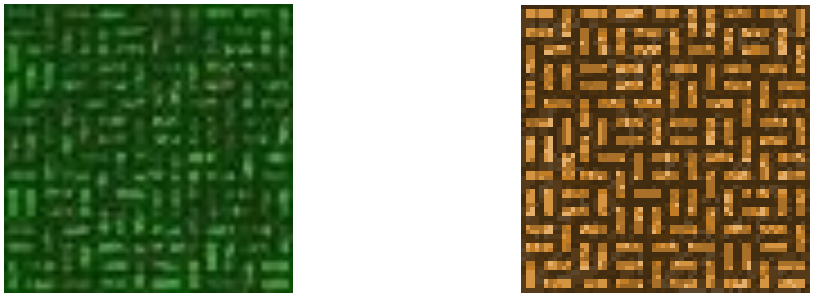
```
private void GenerateTexture()
    {

        List<Vector3> verticies = new List<Vector3>();
        List<int> triangles = new List<int>();
        List<Vector2> uv = new List<Vector2>();

        int blockCount = 0;

        for (int x = 0; x < GlobalVariables.TILE_PER_CHUNK_AXIS; x++)
        {
            for (int y = 0; y < GlobalVariables.TILE_PER_CHUNK_AXIS; y++)
            {
                int id = (int)_tilesDataMap[x, y].GetTileType();

                //Create the verticies
                float xCoord = (x) * GlobalVariables.TILE_SIZE;
                float yCoord = (y) * GlobalVariables.TILE_SIZE;

                verticies.Add(
                    new Vector3(xCoord, yCoord + GlobalVariables.TILE_SIZE, 0));
                verticies.Add(
                    new Vector3(xCoord + GlobalVariables.TILE_SIZE,
                                yCoord + GlobalVariables.TILE_SIZE, 0));
                verticies.Add(
                    new Vector3(xCoord + GlobalVariables.TILE_SIZE, yCoord, 0));
                verticies.Add(
                    new Vector3(xCoord, yCoord, 0));

                //Make a guideline for the building on the triangles
                triangles.Add(0 + blockCount * 4);
                triangles.Add(1 + blockCount * 4);
                triangles.Add(3 + blockCount * 4);
                triangles.Add(1 + blockCount * 4);
                triangles.Add(2 + blockCount * 4);
                triangles.Add(3 + blockCount * 4);


                //Select the right texture for the tile based on the tile id
                float unit = 1f / _spriteSheetSize;

                float idToSpriteSheetSize = (float)id / _spriteSheetSize;
                float xUnit = idToSpriteSheetSize - (int)idToSpriteSheetSize;
                float yUnit = (int)idToSpriteSheetSize * unit;

                uv.Add(new Vector2(xUnit, yUnit));
                uv.Add(new Vector2(xUnit + unit, yUnit));
                uv.Add(new Vector2(xUnit + unit, yUnit + unit));
                uv.Add(new Vector2(xUnit, yUnit + unit));

                blockCount++;
            }
        }

        //Initialize mesh based on the tiles data
        _mesh.Clear();
        _mesh.vertices = verticies.ToArray();
        _mesh.triangles = triangles.ToArray();
```

Фунцкия 3-5. GenerateTexture()

3.1.3 Видове плочки

3.1.3.1 Биом

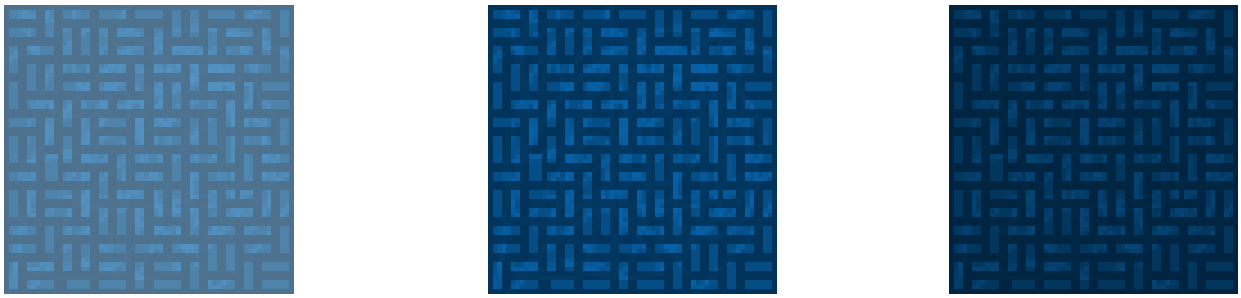
Към текущата версия на играта, съществуват два биома– тревен и саванен (Фигура 3-11). За момента, биомите са чисто козметични и не придават никаква функционалност на съответните плочки.



Фигура 3-11. Два вида биоми – тревен(ляво) и саванен(дясно).

3.1.3.2 Вода

В играта е имплементирана и водна плочка. Играча не може да преминава през този тип плочки. Те от своя страна се делят на три типа– плитки, нормални и дълбоки(Фигура 3-12). Разликата между тях е само козметична.



Фигура 3-12. Плитки(ляво), нормални(среда), дълбоки(дясно)

3.1.3.3 Ресурс

За всяка плочка от някакъв ресурс(желязо, въглища), се създава обект от съответния ресурс с координати съотвестващи на плочката(Фунцкия 3-6). Този обект има случайна ротация и модел. Той съдържа информация за количеството. Когато то достигне 0, обекта се унищожава.

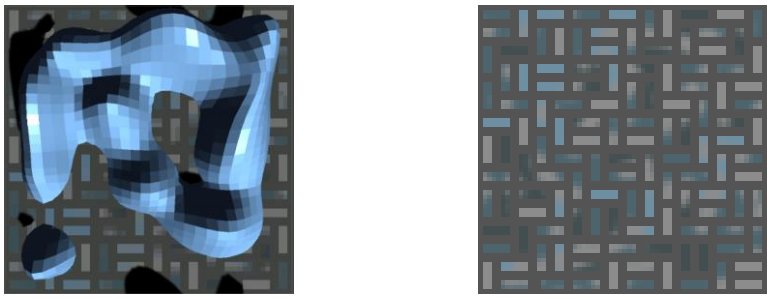
```
private GameObject NewResource(TilesEnum type, int x, int y)
    {
        float xCoord = transform.position.x + (x * GlobalVariables.TILE_SIZE +
GlobalVariables.TILE_SIZE / 2);
        float yCoord = transform.position.z + (y * GlobalVariables.TILE_SIZE +
GlobalVariables.TILE_SIZE / 2);
        int modelRotation = 90 * Random.Range(-4, 4);
        GameObject instance = null;
        switch(type)
        {
            case TilesEnum.IRON_ORE:
                instance = Instantiate(
                _ironOrePrefabs[Random.Range(0, _ironOrePrefabs.Count)],
                new Vector3(xCoord, -2f, yCoord),
                new Quaternion(0f, modelRotation, 0f, 0f));
                break;
            case TilesEnum.COAL:
                instance = Instantiate(
                _coalPrefabs[Random.Range(0, _coalPrefabs.Count)],
                new Vector3(xCoord, -2f, yCoord),
                new Quaternion(0f, modelRotation, 0f, 0f));
                break;
        }

        if (instance != null)
        {
            instance.transform.SetParent(transform.Find("Resources"));
        }
        return instance;
    }
```

Функция 3-6. NewResource().

Когато героя на играча е близо до ресурса, играча има възможност да добива ресурса. Добиването, намалява количеството в обекта и добавя в инвентара на играча.

3.1.3.3.1 Желязо

Желязната плочка (Фигура 3-13) съдържа един от основните ресурси на играта– желязото. Този ресурс се използва за създаване на машините.



Фигура 3-13. Желязна плочка: С руда(ляво) и без руда(дясно)

3.1.3.3.2 Въглища

Въглищата(Фигура 3-14) са ресурс- гориво, което някои машини използват.

Фигура 3-14. Въглищна плочка: С въглища(ляво) и без руда(дясно)

3.2 Инвентар и артикули

В играта е имплементирана система за съхранение и достъп на артикули-инвентар. Играчът има основен инвентар, където може да съхранява своите предмети в слотове и да ги достъпва с един бутон. Освен това, е имплементиран и хотбар, където играча може да съхранява артикули като например машини, за да може по- лесно да ги достъпва.

3.2.1 Артикули

Артикулите са начинът на играта да представи визуално притежанията на играча. Пример за артикул може да бъде желязна руда. Те се запазват в инвентара. Всеки артикул има няколко атрибута:

- Идентификационен номер
- Име (Пример: Iron Ore, Iron Ingot и т.н.).
- Вид (Пример: Ore, Placable, Fuel и други).
- Иконка, която да се показва в инвентара.

3.2.1.1 Видове

В играта са имплементирани няколко вида артикули:

- Гориво
- Руда
- Артикул за поставяне (тези артикули представят машини)
- Други- артикули, които нямат обособена функция

3.2.2 Слот

Както вече беше споменато, инвентарът се състои от определен брой слотове. На вътрешно ниво, слотът представлява контейнер, който съдържа определен брой от даден предмет. На ниво потребителски интерфейс, слотът е квадрат, в който артикула се представя, чрез неговата иконка и количество(Фигура 3-15).

Фигура 3-15. Празен слот(ляво) и слот с желязно кюлче(дясно).

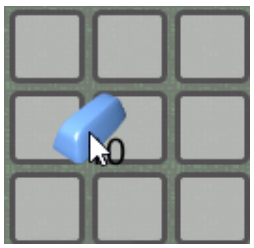
Всеки обект- слот се контролира от прикачен за него скрипт- Slot.cs. Slot.cs наследява функционалността на абстрактния клас SlotAbstract.cs, която позволява като се премести мишката върху слот, той да потъмнее и да изпише името на артикула(Фигура 3-16). Когато мишката се премести върху слота, се създава събитие, което извиква функцията OnPointerEnter(Фунцкия 3-7), за класа SlotAbstract.cs, която сменя цвета на слота и създава панелче с името на артикула, ако има такъв.

Фигура 3-16. Затъмнен слот и панелче с името на артикула.

```
public void OnPointerEnter(PointerEventData eventData)
    {
        if (_item != null)
        {
            _tooltip = Instantiate(
              SerializedGlobalVariables.instance.ItemInfoPanel,
              new Vector3(
                    Input.mousePosition.x,
                    Input.mousePosition.y - GetComponent<RectTransform>().rect.height,
                    0f),
              Quaternion.identity,
              GameObject.FindGameObjectWithTag("Canvas").transform);

            _tooltip.GetComponent<ItemToolTipController>().Initialize(_item.name);
        }
        GetComponent<Image>().color =
               SerializedGlobalVariables.instance.slotOnHoverColor;
    }
```

Фунцкия 3-7. OnPointerEnter()

Slot.cs добавя функционалност за месетене на артикули (Фигура 3-17), като текущо движеният стак, се управлява от скрипта DragContainer.cs.

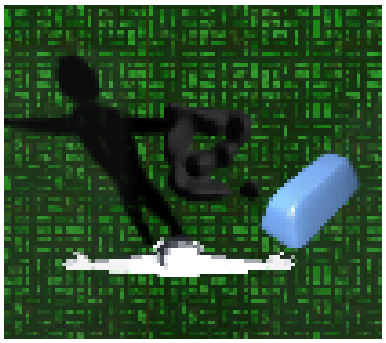


Фигура 3-17. Местене на артикул из инвентара.

Местенето на артикул се извършва, като играчът натисне с мишката върху него. Ако играчът натисне с ляв бутон, взима цялото количество от слота, а ако натисне с десен бутон- само половината. След това играчът мести стакът от артикули. Ако пожелае, играчът може да постави носените артикули в друг слот. Съществуват следните възможности:

- Ако играчът натисне с ляв бутон върху празен слот, ще постави целият стак от артикули в него. Ако натисне с десен- само един артикул от стака ще бъде поставен.
- Ако играчът натисне с ляв бутон върху слот в който вече има същият артикул, ще направи опит да постави целият стак от артикули в него. Ако обаче, не може да се постави целият, защото полученото количество ще бъде по- голямо от максималното за стак, тогава ще бъдат поставени колкото е възможно. Ако е цъкнато с десен бутон и е възможно, ще бъде поставен само един артикул от целият стак. Ако играчът натисне с колецето на мишката, курсорът ще изчезне и с въртенето на колецето, играчът може да пренесе между слотът и влаченото количество, точно толкова артикули, колкото той иска. Този процес се прекъсва при изчерпване на количеството в стаковете от слота или влаченият артикул или натискане на ляв или десен бутон на мишката.
- Ако играчът натисне с ляв бутон върху слот в който има друг артикул, местата на двата стака ще бъдат разменени, т.е. в слота вече ще седи носеният артикул, а артикулът от слота ще бъде местен от играчът.

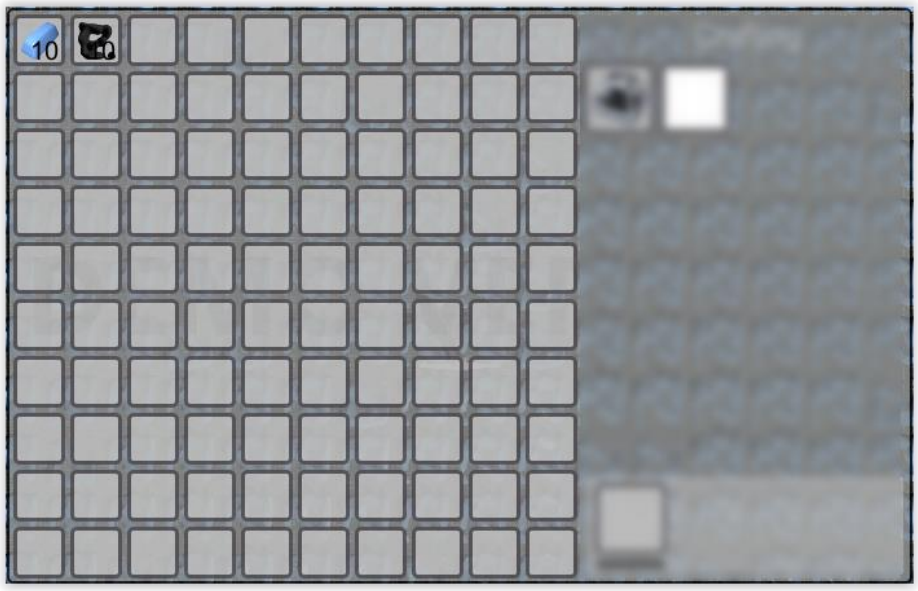
Също така, като допълнителна функционалност е имплементирана възможност за изваждане на артикул от инвентара и оставянето му върху терена на играта (Фигура 3-18). Това се прави, като с влаченият обект се кликне някъде извън инвентара. Обектът създаден при оставянето, се контролира от скрипта Drop.cs, след това играча може да премине и да добави артикула в инвентара си.



Фигура 3-18. Артикули, оставени върху терена на играта.

3.2.3 Основен инвентар на играча

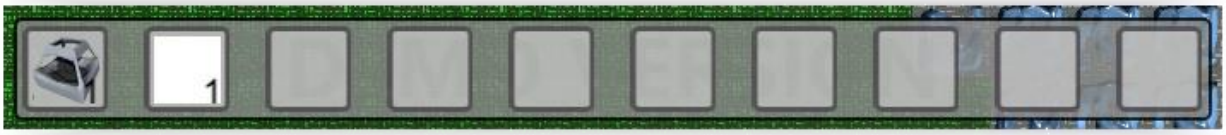
Основния инвентар на играча представлява скрит панел със слотове, като във всеки слот, играча може да съхранява само един тип предмети и до определен брой. Определени са 100 броя слотове в инвентара (Фигура 3-19). Скриптът, който контролира инвентара е PlayerInventoryContoller.cs



Фигура 3-19. Инвентарът на играча.

3.2.4 Хотбар на играча

Хотбарът на играча представлява една лента от 10 слота в долната част на екрана (Фигура 3-20). Той позволява на играчът лесно да достъпва най-важните артикули. Хотбарът се контролира от HotBarController.cs



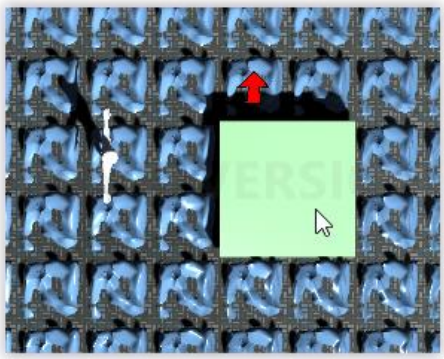
Фигура 3-20. Хотбарът на играча.

3.3 Устройства за добив и обработване на ресурсите

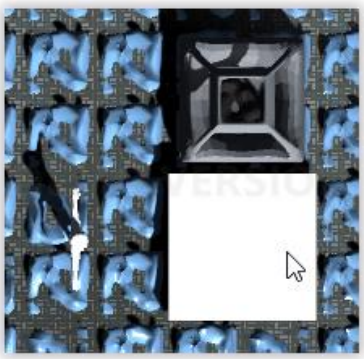
В играта са имплементирани три машини за добив и обработване на ресурсите. Всяка от тях притежава удобен потребителски интерфейс, който да позволява взаимодействие на играча с машината.

3.3.1 Екстрактор на ресурси

Екстракторът на ресурси позволява автоматично добиване на ресурси. Размерът на машината е 2X2. За да работи, трябва да бъде поставена върху поле 2X2 ресурс (желязо, въглища) и да бъде захранена с гориво- въглища. След това, машината започва да работи. При поставянето, се появява стрелка, която е изходът на екстракторът (Фигура 3-21). Ако на нейно място бъде поставена машина за съхранение или обработка на изходния продукт на екстрактора, всички продукти, които екстракторът добива, се изпращат в изходната машина (Фигура 3-22), ако не бъде сложена машина на изхода, изходният продукт отива в изходният слот на екстрактора.

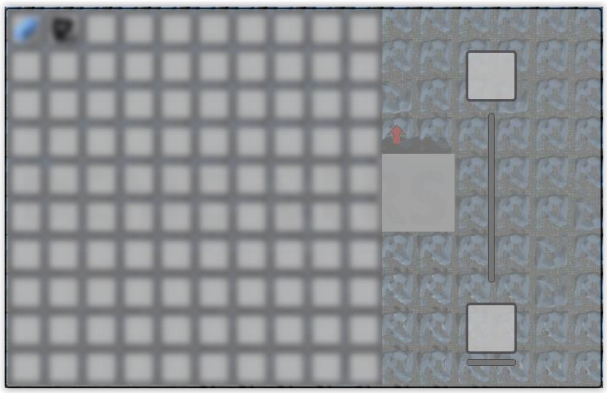


Фигура 3-21. Поставяне на екстракторът.

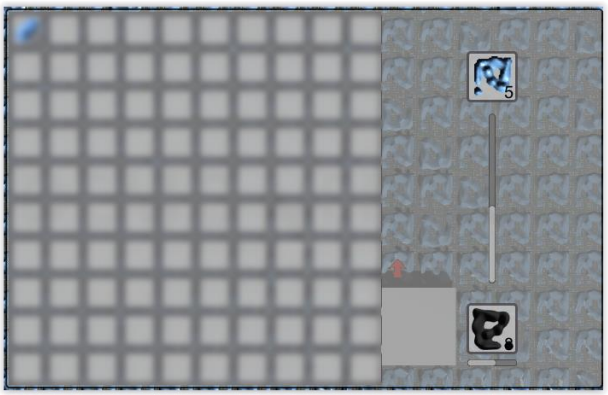


Фигура 3-22. Екстрактор, с машина за топене на руди на изхода му.

Интерфейсът на екстракторът представлява два слота- един за горивото и един за изходният продукт, както и два слайдера- единият показва остатъкът гориво, което в момента се гори, а другият показва прогресът на добив на нов ресурс (Фигура 3-23, Фигура 3-24).



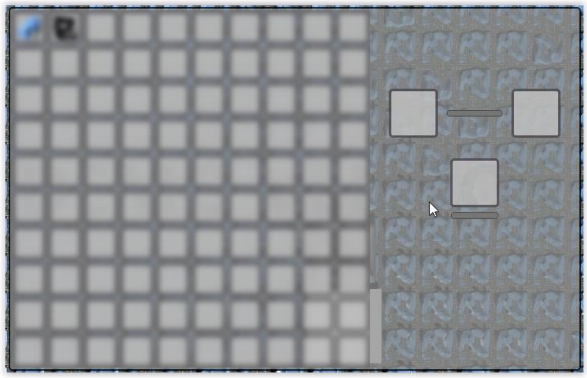
Фигура 3-23. Интерфейсът на екстракторът..



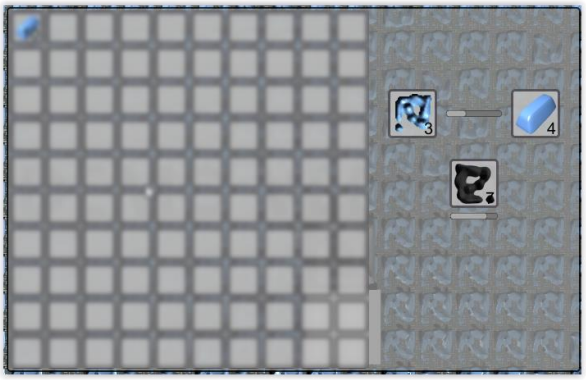
Фигура 3-24. Работещ екстрактор.

3.3.2 Машина за топене на руди

Машината за топене на руди, термично обработва рудите и изходният продукт е кюлче. Тя е с размер 2X2, и може да бъде поставена навсякъде освен върху вода. За да работи, трябва да бъде захранена с гориво- въглища. Интерфейсът представлява три слота- един входен, един изходен и един за горивото, както и два слайдера- единият показва остатъкът гориво, което в момента се гори, а другият показва прогресът на топене на рудата (Фигура 3-25, Фигура 3-26).



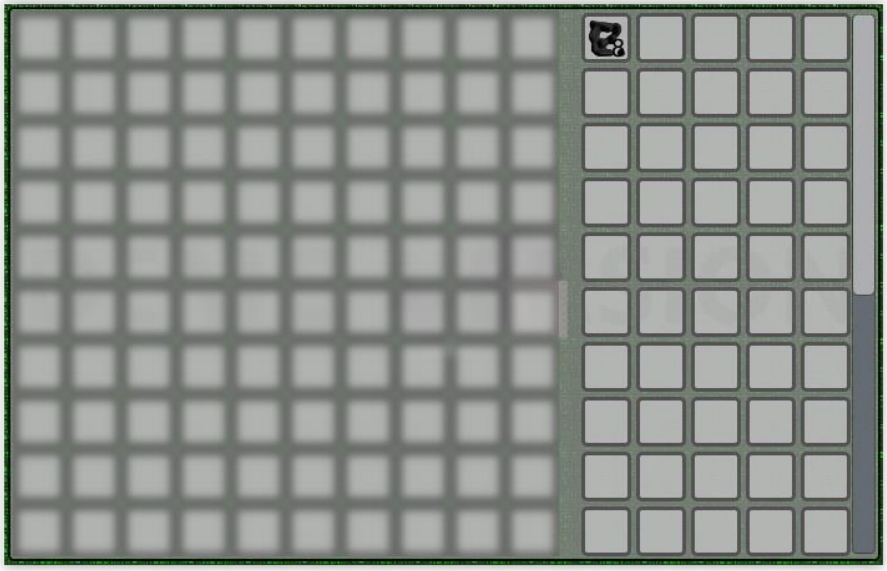
Фигура 3-25. Интефейсът на машината за топене на рудите.



Фигура 3-26. Работеща машина за топене на рудите.

3.3.3 Контейнер за съхранение

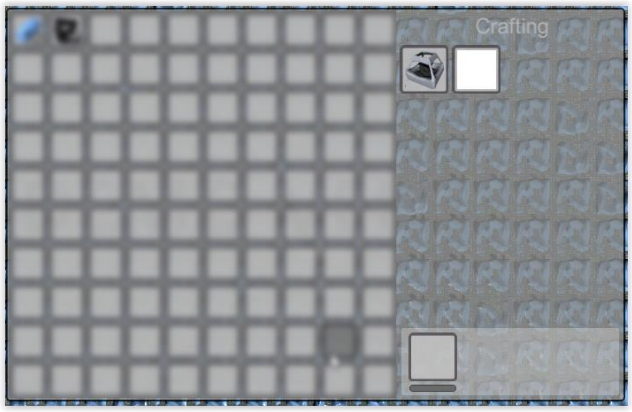
В играта е имплементиран и контейнер за съхранение. Той позволява на играчът да съхранява своите артикули, и лесно да ги достъпва след това. Капацитета на контейнера е 100 слота. Интефейсът на контейнера се състои от слотовете, както и скролбар, за навигация между тях (Фигура 3-27).



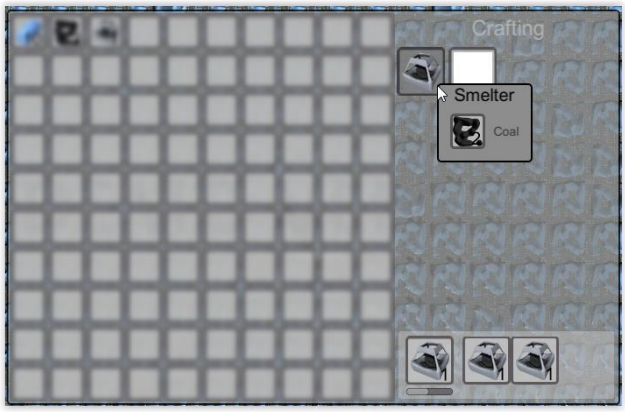
Фигура 3-27. Интерфейсът на контейнера за съхранение.

3.4 Създаване на устройства за добив и обработка на ресурсите

За създаването на устройства за добив и обработка на ресурсите, се грижи интерфейс, който съдържа слотове с машини, които играчът може да създаде и опашка, която показва текущо създаващите се машини (Фигура 3-28). Когато играчът премести своя курсор върху слот с машина, се показват необходимите артикули за нейното създаване. Създаването на избрана машина се извършва за определно време и прогресът към създаването й се показва в слайдера на опашката (Фигура 3-29).



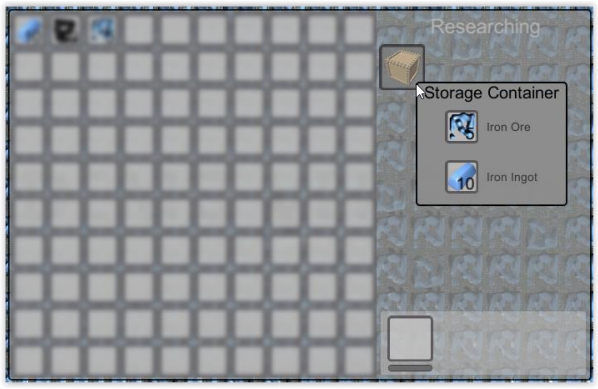
Фигура 3-28. Интерфейсът на създаване на нови машини.



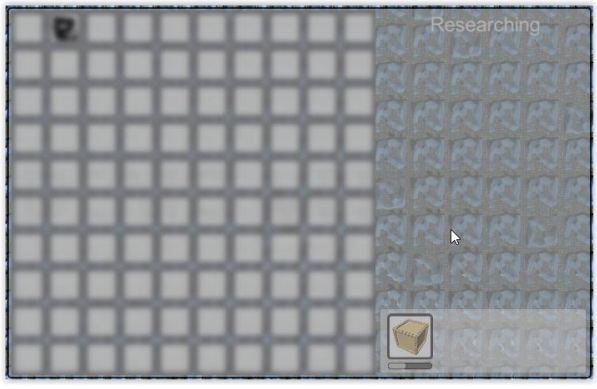
Фигура 3-29. Процес на създаване на артикул.

3.5 Система за отключване на нови устройства

Системата за отключване на нови устройства представлява интерфейс, който се състои от слотове с проучвания, които потребителят да направи. Всяко проучване има необходими артикули за изпълнение, които играчът може да види, когато премести курсора си върху проучването (Фигура 3-30). Има и опашка на текущите проучвания и слайдер който показва прогреса към текущото проучване (Фигура 3-31).

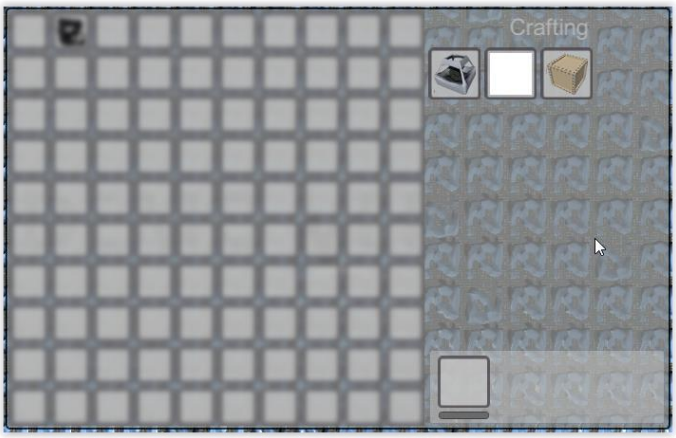


Фигура 3-30. Интерфейсът на системата за отключване на нови устройства.



Фигура 3-31. Процес на отключване на ново устройство.

След като проучването бъде направено, нова машина се добавя като възможност за създаване в системата за създаване на машини (Фигура 3-32).



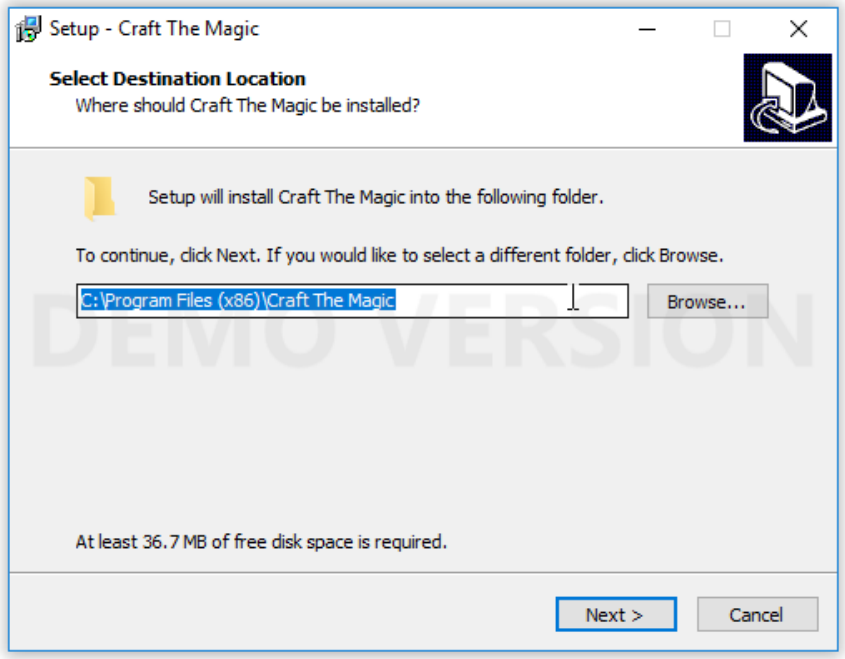
Фигура 3-32. Нов контейнер за съхранение, след като е проучен.

Глава 4. Ръководство на потребителя

4.1 Инсталиране на продукта

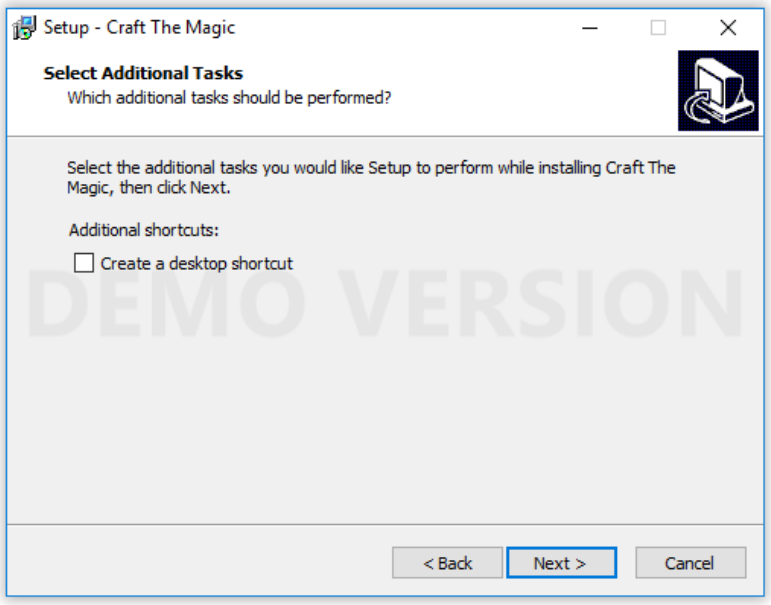
За инсталиране на играта, се използва файлът **setup.exe**.

1. След стартиране на файлът, се отваря прозорец, в който потребителят избира къде да бъде инсталирана играта. По подразбиране е избрана папката C:\Program Files (x86)\ (Фигура 4-1).



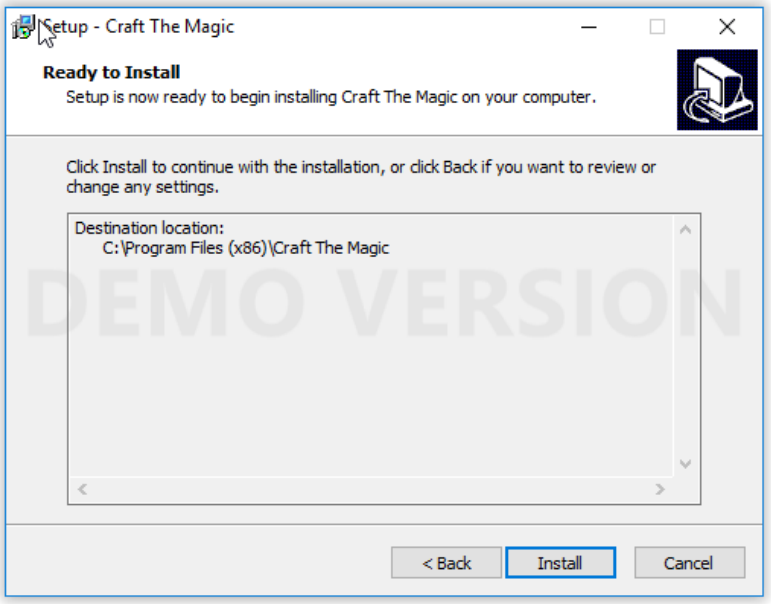
Фигура 4-1. Избор на инсталационна папка.

2. След избор на инсталационна папка и натискане на бутона **Next**, потребителят има възможност да избере дали инсталаторът да създаде пряк път на играта на Работният плот (Фигура 4-2).



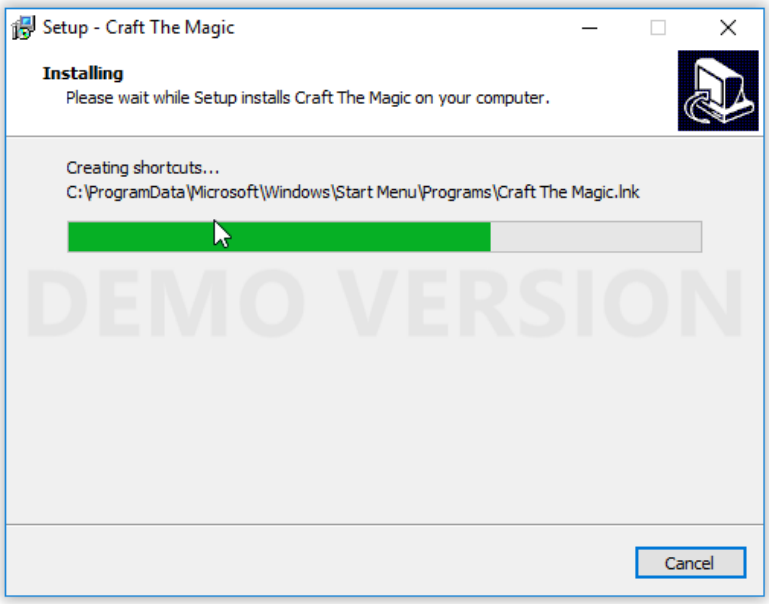
Фигура 4-2. Опция за създаване на пряк път на Работния плот.

3. След избор и натискане на бутона **Next**, следва прозорец, който обобщава настройките на инсталацията, потребителят може да се върне назад, чрез бутона **Back**, за да промени настройките, или да натисне **Install** и да започне финалният етап от инсталацията (Фигура 4-3).



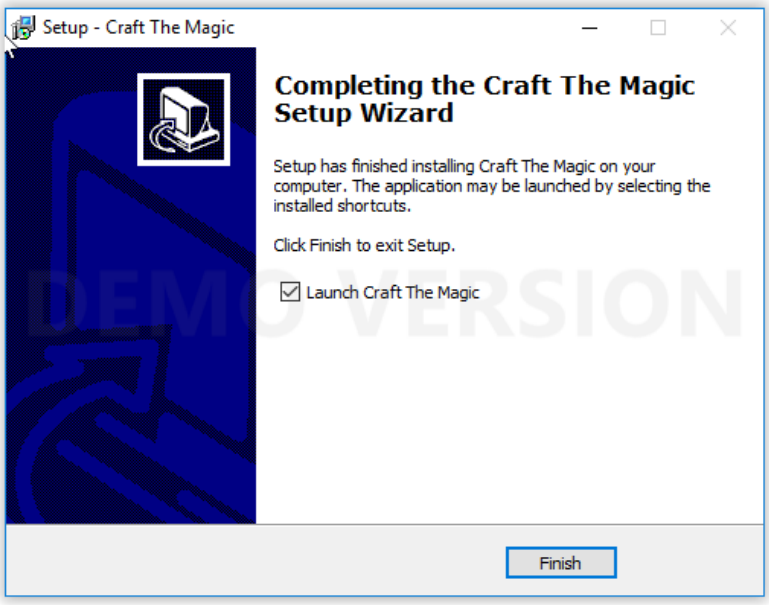
Фигура 4-3. Обобщение на избраните настройки.

4. При натискане на **Install**, в следващият прозорец е отразен прогресът на инсталацията (Фигура 4-4).



Фигура 4-4. Прогресът на инсталацията.

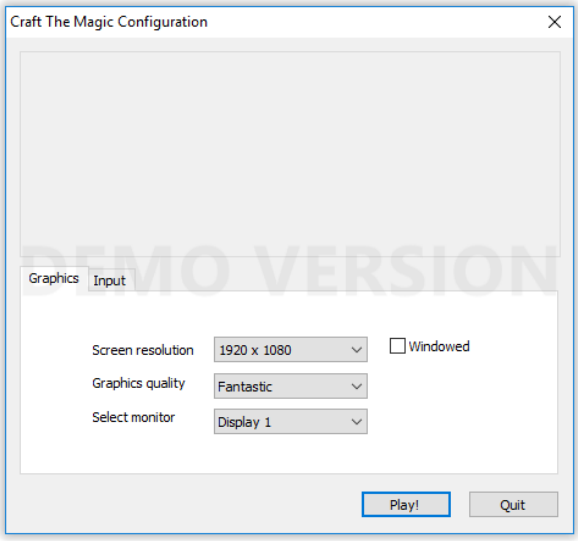
5. След като инсталацията приключи и потребителят натисне бутонът **Next**, се появява възможност, играта да се стартира, след като инсталаторът бъде затворен, чрез бутонът **Finish** (Фигура 4-5).



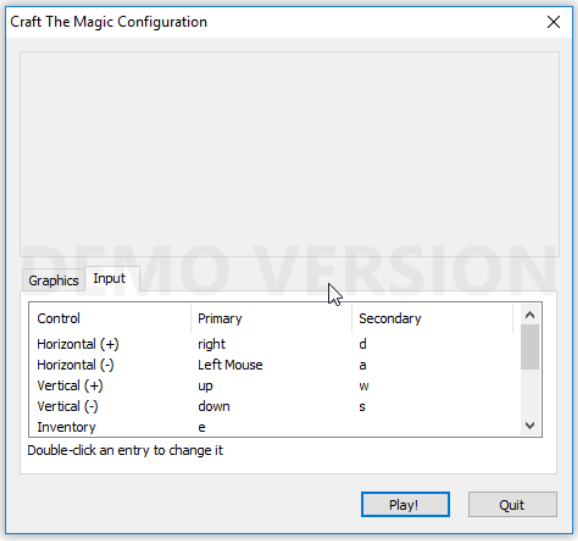
Фигура 4-5. Финален прозорец на инсталацията.

4.2 Стартиране на играта

Стартирането на играта се случва, чрез отваряне на файлът **Craft The Magic.exe**. При стартирането, се появява прозорец, в който потребителят може да настрои своите графични настройки, както и контролите на играта (Фигура 4-6, Фигура 4-7).



Фигура 4-6. Графични настройки.



Фигура 4-7. Настройки на контролите.

4.3 Инструкции за игра

След като играта бъде стартирана, се визуализират света и героя в нов прозорец. Потребителят вече може да контролира своя герой и да изпълнява дейности свързани с играта.

По подразбиране, контролите на играта са настроени по следният начин:

- W или стрелка нагоре- за движение на героя нагоре.
- S или стрелка надолу- за движение на героя надолу.
- A или стрелка наляво- за движение на героя наляво.
- D или стрелка надясно- за движение на героя надясно.

- E- за отваряне на инвентара и системата за създаване на нови устройства
- T- за отваряне на инвентара и системата за отключване на нови устройства
- F- за взимане на артикул оставен върху терена на играта
- Чрез колелцето на мишката, играчът може да контролира мащаба на камерата
- Отварянето на машини, както и добиването на ресурси се случва чрез десният бутон на мишката.
- В инвентара, управлението на слотовете с артикули, се случва чрез левият и десният бутон на мишката, както и колелцето на мишката.
- Хотбарът на играча се управлява, чрез бутоните 1, 2, 3, 4, 5, 6, 7, 8, 9, 0 на клавиатурата.
- Esc се използва за затваряне на инвентар- на играчът или на машините.

Играта се изключва чрез натискане на бутоните **Alt** и **F4**.

4.4 Деинсталация на продукта

Тъй, като при инсталацията на играта не се създават никакви системни файлове, за деинсталация на продукта е достатъчно само да се изтрият файловете създадени при инсталирането- **Craft The Magic_Data** и **Craft The Magic.exe**.

Заключение

След процес на проучване на подходящ алгоритъм за процедурно генериране на терен, създаване на машини, артикули и начин на подходяща имплементация, крайният продукт отговаря на всички поставени цели и задачи. Потребителските интерфейси са лесно достъпни и лесно разбираеми за всеки, а имплементираните машини дават възможност за оптимизация и автоматизация на логистическата верига. Играчът може лесно да управлява своите ресурси и да ги съхранява.

В следствие на вече имплементирани основни машини за автоматизация и съхранение на ресурси и артикули, бъдещото развитие на играта включва имплементиране на набор от нови и по- сложни машини. Наред с тези нови машини, ще бъдат имплеменитрани нови и разнотипни ресурси.

Относно различните биоми, ще бъдат имплементирани обекти свързани с околната среда- растения и други, и те ще бъдат тематични спрямо всеки биом.

Тъй, като, освен препятствие, водата не притежава друга функционалност, в бъдещият процес на разработка, тя ще придобие различна функционалност на ресурс.

Само създаването на машини, които да образуват автономна логистическа мрежа, представят твърде линейно развитие на играта, водеща до един край- създадена база, развита до всевъзможни нива. За да се добавят възможни изходи на играта, ще бъдат имплементирани врагове, които значително да променят игралният процес на играчът и да представят предизвикателство в лицето на постоянно развиващи се вражески бази.

Използвана литература

1. Biagioli, A. (2018, 02 24). *Understanding Perlin Noise*. Retrieved from Adrian's soapbox: http://flafla2.github.io/2014/08/09/perlinnoise.html
2. *Campaign - First steps*. (2018, 02 14). Retrieved from Factorio Wiki: https://wiki.factorio.com/Campaign#First_steps
3. *Campaign - New Hope*. (2018, 02 14). Retrieved from Factorio Wiki: https://wiki.factorio.com/Campaign#New_hope
4. *CryEngine*. (2018, 02 16). Retrieved from Wikipedia: https://en.wikipedia.org/wiki/CryEngine
5. *History of video games*. (2018, 02 23). Retrieved from Wikipedia: https://en.wikipedia.org/wiki/History_of_video_games
6. *Home page*. (2018, 02 14). Retrieved from Factorio: https://www.factorio.com/
7. *Infinifactory*. (2018, 02 14). Retrieved from Zachtronics: http://www.zachtronics.com/infinifactory/
8. *Learning how Perlin noise works*. (2018, 02 25). Retrieved from Huttar: http://www.huttar.net/lars-kathy/graphics/perlin-noise/perlin-noise.html
9. *Ottawa Citizen.* (1975, 10 08). Retrieved from Google Newspapers: https://news.google.com/newspapers?id=rKYyAAAAIBAJ&sjid=pe0FAAAAIBAJ&pg=916,3790974&dq=josef-kates&hl=en
10. *Unity (Game Engine)*. (2018, 02 14). Retrieved from Wikipedia: https://en.wikipedia.org/wiki/Unity_(game_engine)
11. *Unreal Engine*. (2018, 02 16). Retrieved from Wikipedia: https://en.wikipedia.org/wiki/Unreal_Engine
12. *Vertex (geometry)*. (2018, 02 12). Retrieved from Wikipedia: https://en.wikipedia.org/wiki/Vertex_(geometry)

Фигури

Функции

Съдържание