**RĪGAS VALSTS TEHNIKUMS**

DATORIKAS NODAĻA

Izglītības programma: Programmēšana

**KVALIFIKĀCIJAS DARBS**

**“Stratēģijas datorspēle par viduslaiku ekonomiku un tirdzniecību”**

Paskaidrojošais raksts 135 lpp.

Audzēknis: Kārlis Egle

Prakses vadītājs: Ilona Demčenko

Nodaļas vadītājs: Normunds Barbāns

**Rīga 2024**

**ANOTĀCIJA**

Kvalifikācijas darbs "Stratēģijas datorspēle par viduslaiku ekonomiku un tirdzniecību" ir izstrādāts Rīgas Valsts tehnikuma Datorikas nodaļā. Projekta mērķis bija radīt stratēģijas datorspēli, kas precīzi atspoguļo viduslaiku ekonomikas un tirdzniecības mehānismus. Spēle piedāvā spēlētājiem iespēju iejusties viduslaiku tirgotāju, feodālo zemju valdnieku un amatnieku lomās, piedzīvojot vēsturiskā perioda ekonomiskos izaicinājumus un iespējas. Spēles izstrādei tika izmantotas mūsdienīgas programmēšanas un dizaina metodes, kā arī Unity platforma un C# programmēšanas valoda. Spēle ir veidota 2D grafikas stilā, un tā darbojas uz Windows operētājsistēmas.

Šis darbs ietver ievadu, uzdevuma nostādni, prasību specifikācīju, uzdevuma risināšanas līdzekļu izvēles pamatojumu, programmatūras produkta modelēšanu un projektēšanu, datu struktūru aprakstu, lietotāja ceļvedi, nobeigumu un pielikumus. Uzdevumu nostādnē ir norādīti uzdevumi, kurus sistēmai būs nepieciešams veikt. Prasību specifikācija sastāv no ieejas un izejas informācijas, kā arī no sistēmas funkcionālajām un nefunkcionālajām prasībām. Uzdevuma risināšanas līdzekļu izvēles pamatojumā ir norādīti, kādi līdzekļi tiks izmantoti izstrādāšanai un kādiem nolūkiem tie tiek izmantoti. Programmatūras produkta modelēšanas un projektēšanas apraksts sastāv no sistēmas struktūras modeļa, kas ietver sistēmas arhitektūru un ER modeli, un funkcionālās sistēmas modeļa, kas satur datu plūsmas modeli. Datu struktūru aprakstā tiek parādīta datu bāzes relāciju shēma, kā arī tabulu struktūra ar aprakstu, kas ietver datu tipu un datu garumu norādīšanu. Lietotāja ceļvedī ir norādītas nepieciešamās sistēmas prasības aparatūrai un programmatūrai, kā arī sistēmas instalācija un palaišana, kā arī programmas apraksts, kas paskaidro, kā pareizi lietot sistēmu.

Kvalifikācijas darbs sastāv no 135 lapaspusēm, kurās ietilpst 36 attēli, 4 tabulas un 1 pielikums “Programmas pirmteksts”.

**ANNOTATION**

The qualification paper "Strategy Computer Game on Medieval Economy and Trade" was developed at the Riga State Technical School's Department of Computer Science. The goal of the project was to create a strategy computer game that accurately reflects the mechanisms of medieval economy and trade. The game offers players the opportunity to immerse themselves in the roles of medieval merchants, feudal lords, and craftsmen, experiencing the economic challenges and opportunities of the historical period. Modern programming and design methods were used in the development of the game, along with the Unity platform and the C# programming language. The game is designed in a 2D graphic style and operates on the Windows operating system.

This work includes an introduction, task statement, requirement specification, justification for the choice of task-solving tools, modeling and design of the software product, description of data structures, user guide, conclusion, and appendices. The task statement specifies the tasks the system needs to perform. The requirement specification consists of input and output information, as well as the system's functional and non-functional requirements. The justification for the choice of task-solving tools indicates what tools will be used for development and for what purposes they are used. The description of the modeling and design of the software product consists of the system structure model, which includes the system architecture and the ER model, and the functional system model, which contains the data flow model. The description of data structures shows the database relational schema, as well as the table structure with a description, including data type and data length indications. The user guide specifies the necessary hardware and software requirements for the system, as well as system installation and launch, along with a program description explaining how to use the system correctly.

The qualification paper consists of 135 pages, including 36 images, 4 tables, and 1 attachment.

**SATURS**

[**IEVADS 4**](#_ci6e8911bk67)

[**1. UZDEVUMA NOSTĀDNE 5**](#_m2e9bvwsby7)

[**2. PRASĪBU SPECIFIKĀCIJA 6**](#_55xjxyk9s20l)

[2.1. Ieejas un izejas informācijas apraksts 6](#_lkmpqe4o6hk9)

[2.1.1. Ieejas informācijas apraksts 6](#_rc6be9psb0v7)

[2.1.2. Izejas informācijas apraksts 6](#_mx11mh4djcgg)

[2.2. Funkcionālās prasības 7](#_vtp4kiykywq3)

[2.3 Nefunkcionālās prasības 7](#_1kciyyw3c0q3)

[**3. UZDEVUMA RISINĀŠANAS LĪDZEKĻU IZVĒLES PAMATOJUMS 11**](#_g8mes85j7p1h)

[**4. PROGRAMMATŪRAS PRODUKTA MODELĒŠANA UN PROJEKTĒŠANA 12**](#_48se0mj7kufm)

[4.1. Sistēmas struktūras modelis 12](#_3znysh7)

[4.1.1. Sistēmas arhitektūra 12](#_9rz61axfez2f)

[4.1.1. Sistēmas ER modelis 13](#_tis2xul1etea)

[4.2. Funkcionālais sistēmas modelis 14](#_wukjmsebg0gh)

[4.2.1. Datu plūsmu modelis 14](#_rxkddc3p0clo)

[**5. DATU STRUKTŪRAS APRAKSTS 18**](#_hurutj6kk3tj)

[**6. LIETOTĀJA CEĻVEDIS 19**](#_rvvuaxccfoqo)

[6.1. Sistēmas prasības aparatūrai un programmatūrai 20](#_yoqjlueaadnc)

[6.2. Sistēmas instalācija un palaišana 20](#_l9wqtkjwm9s7)

[6.3. Programmas apraksts 21](#_h694mvrfqgg8)

[6.4. Testa Piemērs 36](#_tssuvhluveb)

[**NOBEIGUMS 38**](#_ob572q4war29)

[**INFORMĀCIJAS AVOTI 39**](#_1ksv4uv)

# IEVADS

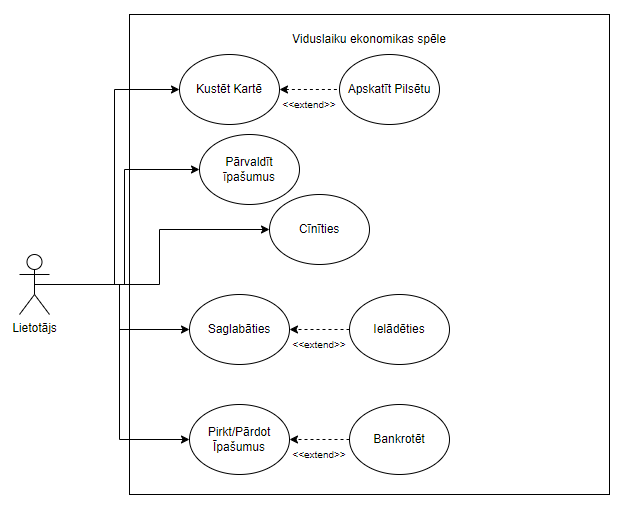
Datorspēļu ir daudz, bet to kvalitāte ir ļoti variēta, un bieži vien par tām tiek prasītas absurdas cenas, kas nav to vērtas. Tādēļ izvēlējos radīt pats savu spēli, kuru varēšu noslīpēt līdz saviem standartiem un modificēt to, lai spētu spēli izbaudīt ilglaicīgi. Šī spēle ir domāta jebkuram stratēģījas spēļu etuziastam, kas vēlas spēli, kurā var iedziļināties un pazust pāris stundas. Tā būš grūta iesācējiem, bet kad tiks apgūta, radīs lielu gandarījumu spēlētājam un arīliks padomāt.

Šī kvalifikācijas darba mērķis ir izstrādāt stratēģijas datorspēli, kas precīzi atspoguļo viduslaiku ekonomikas un tirdzniecības mehānismus. Spēle tiks veidota tā, lai spēlētāji varētu iejusties viduslaiku tirgotāju, feodālo zemju valdnieku un amatnieku lomās, piedzīvojot šī vēsturiskā perioda ekonomiskos izaicinājumus un iespējas. Spēlētājiem būs jāpieņem stratēģiski lēmumi, pārvaldot resursus, attīstot tirdzniecības ceļus un mijiedarbojoties ar spēles vidi.

# UZDEVUMA NOSTĀDNE

Kvalifikācijas darba uzdevums ir izveidot stratēģijas datorspēli datoram par viduslaiku ekonomiku un tirdzniecību. Sistēmā nepieciešams realizēt iespēju ietekmēt un piedalīties realistiskā viduslaiku ekonomikas simulācijā. Tieši šādu spēli nav vēl neviens izveidojis, tāpēc tā ir diezgan unikāla. To spēlēt varēs jebkurš stratēģijas žanra entuziasts. Lietotājam būs iespējas veikt šādas funkcijas:

* Kustēt kartē, kurā var arī apskatīt pilsētas
* Pārvaldīt īpašumus
* Cīnīties ar pretiniekiem
* Saglabāt progresu un iēlādēt iepriekš saglabāto
* Pirkt un pārdot īpašumus.



1.1.att. Lietojumgadījuma diagramma

# 2. PRASĪBU SPECIFIKĀCIJA

## 2.1. Ieejas un izejas informācijas apraksts

### 2.1.1. Ieejas informācijas apraksts

Sistēmā tiks nodrošināta šādas ieejas informācijas apstrāde.

1. Informācija par **spēlētāju** sastāvēs no šādiem datem:

* Prasmes - spēlētāja iegūto prasmju un to līmeņu saraksts - saraksts (piem. [“Izlūkošana”:”3”, “Jāšana”:”4”] ).
* Atribūti - spēlētāja atribūtū līmeņu saraksts - saraksts (piem. [2,4,2,1,3]).
* Īpašumi - spēlētāja īpašumu saraksts - saraksts (piem. [“Banka”, “Ferma”]).
* Lietas - spēlētāja lietu un to daudzuma saraksts - saraksts (piem. [“Zobeni”:”2”, “Ogles”:”5”]).
* Resursi - spēlētāja resursu daudzuma saraksts - saraksts (piem. [“150”, “9”]).

2. Informācija par **laukumu** sastāvēs no šādiem datem:

* Tips - laukuma tips - skaitlis līdz 5 rakstzīmēm (piem. “43”).

3. Informācija par **pilsētu** sastāvēs no šādiem datem:

* Īpašumi - pilsētas īpašumu saraksts - saraksts (piem. [“Banka”, “Ferma”]).
* Lietas - pilsētas lietu un to daudzuma saraksts - saraksts (piem. [“Zobeni”:”2”, “Ogles”:”5”]).
* Resursi - pilsētas resursu daudzuma saraksts - saraksts (piem. [“150”, “9”]).

### 2.1.2. Izejas informācijas apraksts

1. **Spēlētāja kustināšana** pēc attiecīgās pogas ievades pārvieto spēlētāja personāžu ekrānā, ja tas iespējams.
2. **Ieešana pilsētā**, kad spēlētāja personāžs atrodas laukumā, kurš satur pilsētu, un nospiež attiecīgo pogu, lai tajā ieetu, parādas logs, kurā iespējams iegādāties un pārdot lietas un īpašumus attiecīgajā pilsētā.
3. **Ipašumu, lietu, prasmju un atribūtu parvaldīšana**, kad nospiež uz attiecīgās pogas, parādās logs, kurā ir iespējams pārvladīt attiecīgo sadaļu.
4. **Atvērt galveno izvēlni**, kad nospiež attiecīgo pogu atverās galvenās izvēlnes logs, kurā iespējams mainīt iestatījumus, saglabāties, ielādēties un iziet no spēles.

## 2.2. Funkcionālās prasības

1. Jānodrošina personāža kustība.
   1. Kad tiek nospiests attiecīgais tastiņš, jāiet uz laukumu attiecīgajā virzienā.
2. Jānodrošina spēja ieet pilsētā.
   1. Kad personāžs atrodas uz laukuma, kurš satur pilsētu, tiek atvērts pilsētas logs.
   2. Pilsētas logā jābūt iespējamam iegādāties un pārdot lietas un īpašumus.
3. Jānodrošina īpašumu, lietu, prasmju un atribūtu parvaldīšanu.
   1. Kad tiek nospiesta attiecīgā poga vai taustiņš, jāatveras attiecīgajam logam.
   2. Attiecīgajos logs jāspēj pārvaldīt attiecīgo sadaļu.
4. Jānodrošina spēja atvērt galveno izvēlni.
   1. Kad nospiež attiecīgo pogu vai taustiņu, jāatveras izvēlnes logam.
   2. Izvēlnes logā jāspēj pārvaldīt iestatījumus.
   3. Izvēlnes logā jāspēj sagalbāties un ielādēties.
   4. Izvēlnes logā jāspēj aizvērt spēli, ko darot jāparādas brīdinājumam, kurā vaicā vai spēlētājs nevēlas saglabāties pirms spēles aizvēršanas.

## 2.3 Nefunkcionālās prasības

1. Sistēmai jābūt rakstītai C# valodā.
2. Sistēma jāveido Unity platformā.
3. Sistēmai jāstrādā bezsaistē.
4. Grafiskajai daļai jābūt 2D stilā.
5. Sistēmai jāfunkcionē modernā Windows operētājsistēmā.



2.1. att. Kartes skata skice

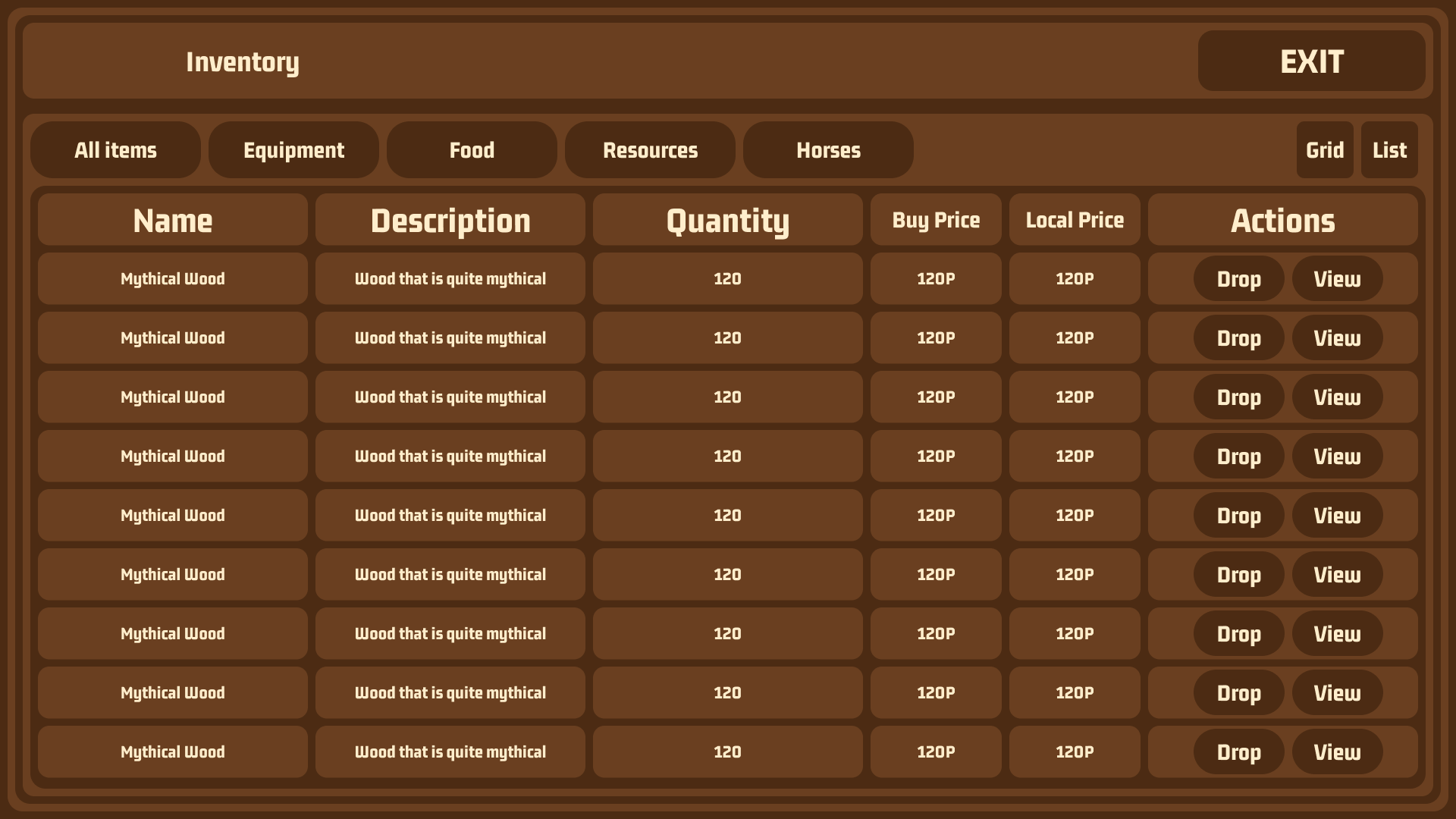
Šī skice attēlo kartes skatu, kas būs primārais sistēmas skats, kurā var kustināt personāžu pa laukumiem, redzēt resursus un nospiest pogas, kas atver attiecīgos logus.

2.2. att. Main Menu skata skice

Šī skice attēlo Main Menu skatu, kas tiek rādīts, kad spēle tiek atvērta, kurā ir iespējams saglabāties, ielādēties, pārvaldīt opcijas, sākt jaunu spēli un aizvērt programmu.

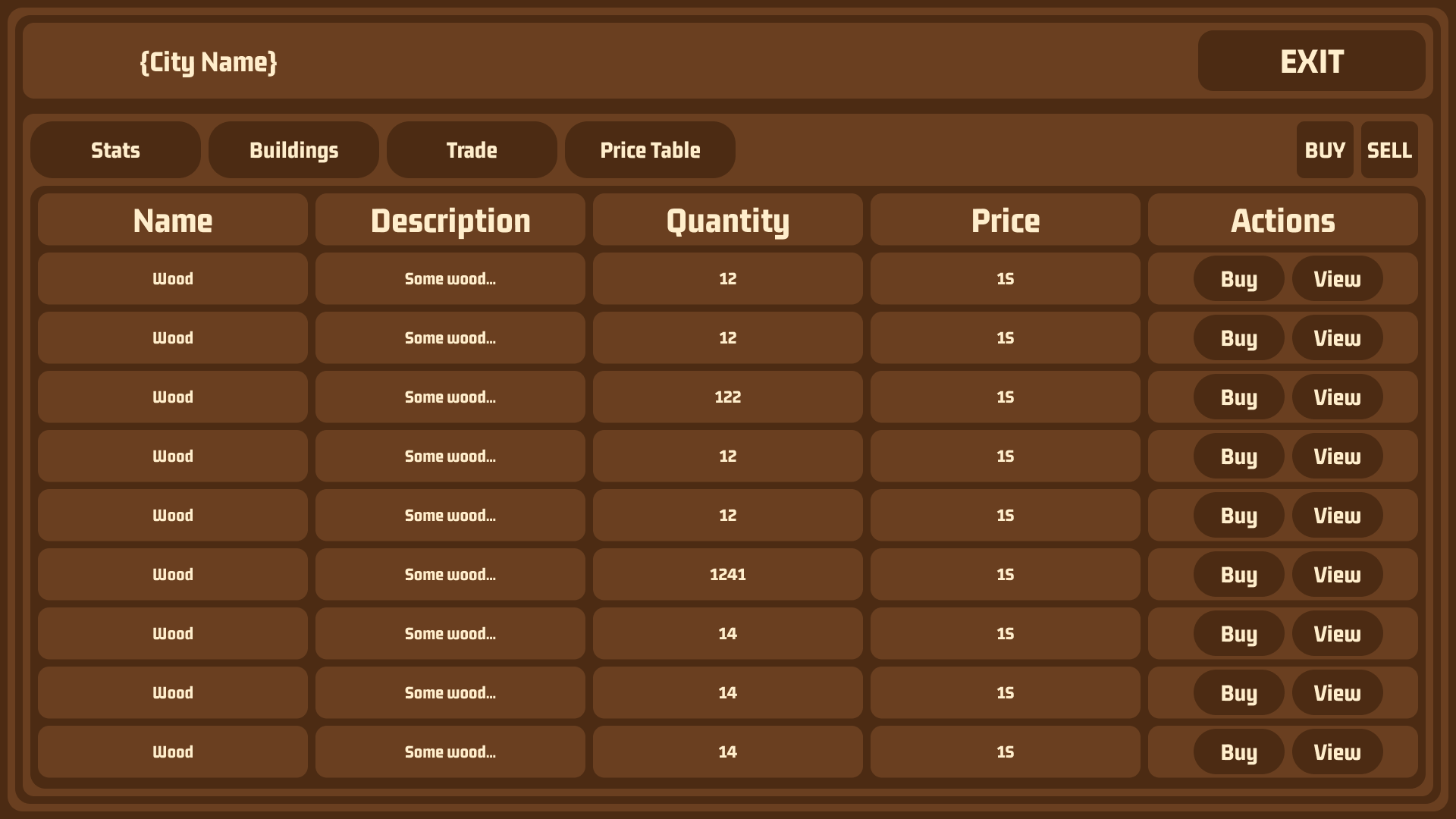
2.3. att. Character Menu skata skice

Šī skice attēlo Character Menu skatu, kurā var pārvaldīt personāža prasmes un atribūtu līmeņus.



2.4. att. Inventory Menu skata skice

Šī skice attēlo Inventory Menu skatu, kurā var redzēt un pārvaldīt personāžam piederošās lietas.



2.5. att. City Menu Trade skata skice

Šī skice attēlo City Menu Trade skatu, kurā var redzēt, pirkt un pārdot lietas pilsētā.

# 

# 3. UZDEVUMA RISINĀŠANAS LĪDZEKĻU IZVĒLES PAMATOJUMS

Šīs datorspēles pilnveidošanai, tika izmantoti vairāki rīki.

Primārajiem spēles aspektiem tika izmantots Unity game engine (2022.3.10F1). Unity ir plaši atzīta spēļu izstrādes platforma, kas piedāvā daudzveidīgas iespējas 2D un 3D spēļu izstrādei. Unity nodrošina spēcīgu rīku komplektu vizuālo elementu ievietošanai un animēšanai. Spēles vizuālie elementi tika rūpīgi integrēti un sasaistīti ar skriptiem, lai izveidotu interaktīvu un saistošu spēles pieredzi.

Skripti tika veidoti C# valodā, izmantojot Visual Studio Community (2022 17.8.3). Visual Studio ir populārs integrētas izstrādes vide (IDE), kas piedāvā ērtu un efektīvu integrāciju ar Unity. Tas nodrošina iespēju ātri rakstīt, testēt un atkļūdot kodu. Visual Studio piedāvā plašu rīku un funkciju klāstu, kas palīdzēja veidot sarežģītus spēles loģikas elementus un nodrošināt vienmērīgu darbību.

Saskarnes dizaina veidošanai tika izmantots Figma (124.0.2). Figma ir efektīvs un populārs bezmaksas dizaina rīks, kas ļauj detalizēti veidot un prototipēt lietotāju saskarnes. Ar Figma palīdzību tika izveidoti spēles lietotāju interfeisa (UI) dizaini, kas pēc tam tika pārnesti uz Unity, lai radītu vizuāli pievilcīgu un lietotājam draudzīgu vidi.

Git(2.45.1) tika izvēlēts priekš versiju kontroles, jo tas ir industrījas standarts šajā jomā.

# 4. PROGRAMMATŪRAS PRODUKTA MODELĒŠANA UN PROJEKTĒŠANA

## Sistēmas struktūras modelis

### Sistēmas arhitektūra

Datorspēles funkcionālā dekompozīcijas diagramma (sk. 4.1. att.) sastāv no 2 apakšsistēmām: spēlētāju datu apstrāde un kartes izmaiņu apstrāde.



4.1. att. Sistēmas funkcionālā dekompozīcijas shēma.

**Spēlētāju datu apstrādes apakšsistēma -** ir atbildīga par visiem spēlētāja datiem un to pārvaldību. Tā ietver Atribūtu un prasmju izmaiņas, Lietu un īpašumu izmaiņas un Kustību veikšanu.

**Kartes izmaiņu apstrādes apakšsistēma -** ir atbildīga par visām izmaiņām spēles kartē - Cenu izmaiņām, Laika izmaiņām un Misiju progresa izmaiņām.

### Sistēmas ER modelis

4.2. att. Sistēmas ER-diagramma

Sistēmas ER-modelis sastāv no 3 entitījām (skat. 4.2. att.), kas nodrošina pamatinformācijas uzglabāšanu un apstrādi. Tās ir **spēlētājs**, kurā ietilpst *prasmes*, *atribūti*, *īpašumi*, *lietas* un *resursi*, **laukums**, kurā ietilpst tā *tips*, un **pilsēta**, kurā ietilpst *lietas*, *resursi* un *īpašumi*.

Starp entitijam **spēlētājs** un **laukums** ir saite “Atrasties” ar relāciju viens pret daudziem, jo viens **spēlētājs** var atrasties vairākos **laukumos** un katrā **laukumā** var atrasties tiaki viens **spēlētājs**.

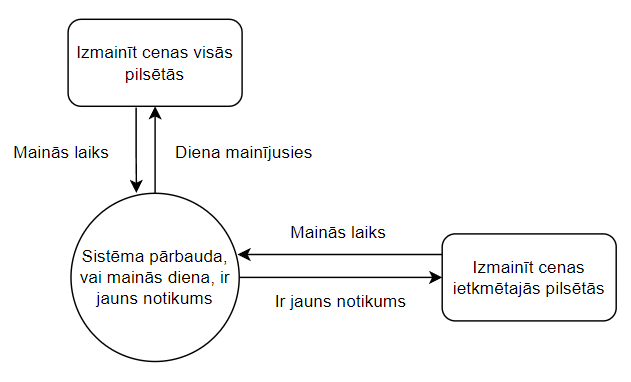
Starp entitijam **laukums** un **pilsēta** ir saite “Saturēt” ar relāciju viens pret vienu, jo **laukumā**  var būt tikai viena **pilsēta,** un katra **pilsēta** var būt tikai vienā **laukumā**.

## Funkcionālais sistēmas modelis

### 4.2.1. Datu plūsmu modelis

Modulis “Izmainīt Īpašumu un Lietu cenas”:

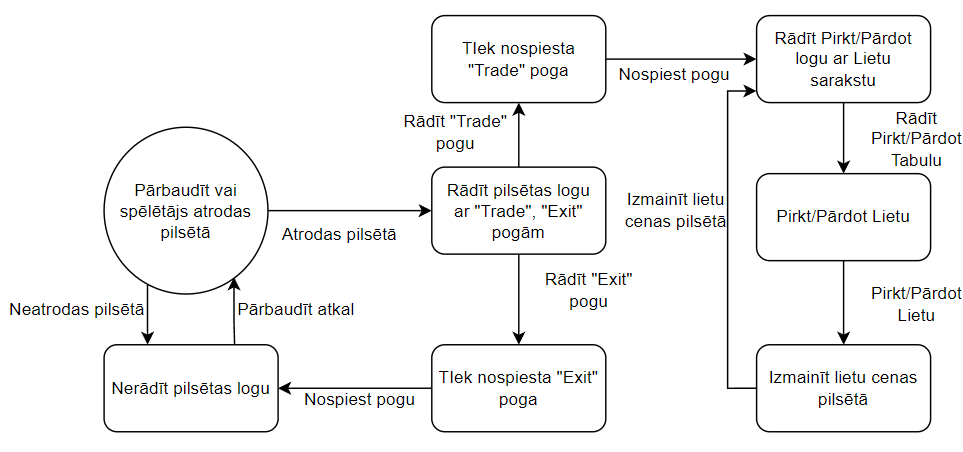
* spēles gaitā, kad mainās laiks, sistēma pārbauda vai mainās diena, vai ir jauns notikums
* ja mainījusies diena, tad visās pilsētās cenas tiek izmainītas
* ja ir jauns notikums, tad tikai tā ietekmētajās pilsētās tiek izmainītas cenas



4.3.att. Modulis “Izmainīt Īpašumu un Lietu cenas”

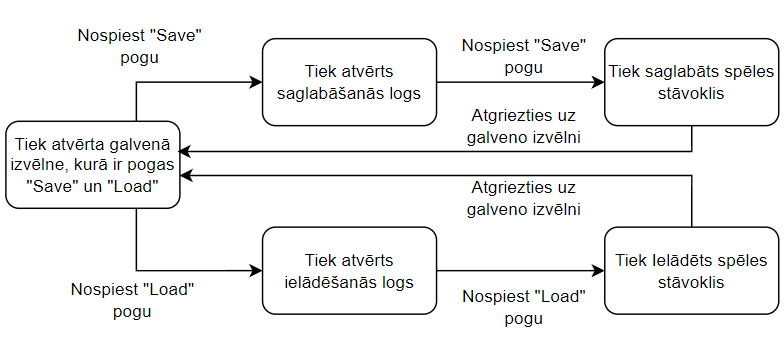
Modulis “Pirkt/Pārdot Īpašumus un Lietas”:

* spēles gaitā, kad spēlētājs kustās, sistēma pārbauda vai tas atroda pilsētā
* ja neatrodas, tad atkal tiek veikta pārbaude
* ja atrodas, tad tiek rādīts pilsētas logas, kas satur “Trade” un “Exit” pogas
* kad tiek nospiesta “Exit” poga, pilsētas logs tiek aizvērts
* kad tiek nospiesta “Trade” poga, tiek parādīta Pirkt/Pārdot tabula
* ja tajā nopērk vai pārdod lietu, tad tiek izmainītas lietu cenas pilsētā

4.4.att. Modulis “Pirkt/Pārdot Īpašumus un Lietas”

Modulis “Saglabāties/Ielādēties”:

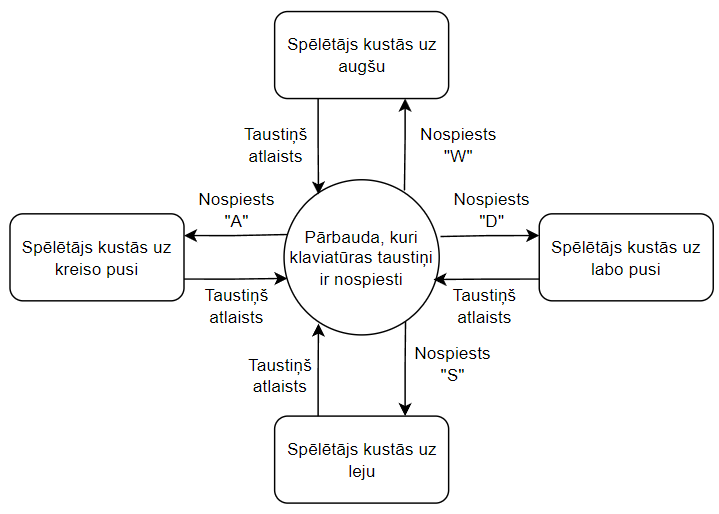
* spēles gaitā, kad spēlētājs atver galveno izvēlni, tajā atrodas “Save” un “Load” pogas.
* ja nospiež “Save” pogu, tad atveras saglabāšanās logs
* ja tajā nospiež “Save” pogu, tad tiek saglabāts spēles stāvoklis, un spēlētājs tiek atgriezts uz galveno izvēlni
* ja nospiež “Load” pogu, tad atveras ielādēšanās logs
* ja tajā nospiež “Load” pogu, tad tiek ielādēts spēles stāvoklis, un spēlētājs tiek atgriezts uz galveno izvēlni



4.5.att. Modulis “Saglabāties/Ielādēties”

Modulis “Kustināt spēlētāju”:

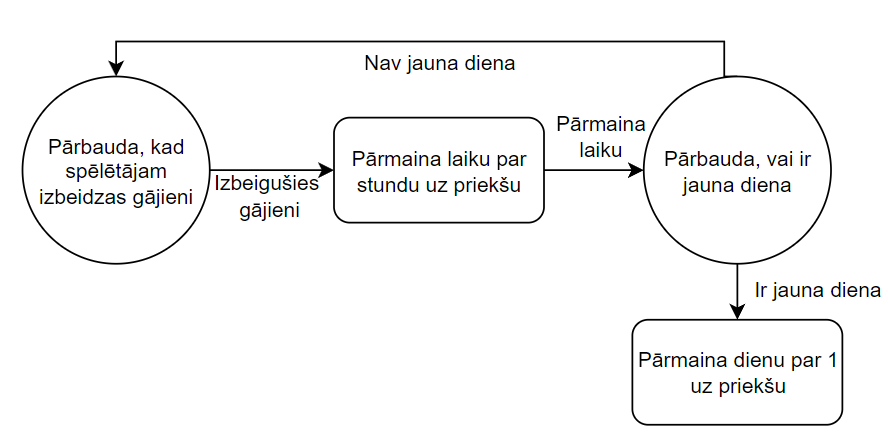
* spēles gaitā, kad spēlētājs nospiež “W”, “A”, “S” vai “D” taustiņu, tas kustās attiecīgi uz augšu, pa kreisi, uz leju vai pa labi.



4.6.att. Modulis “Kustināt spēlētāju”

Modulis “Mainīt Dienu, Laiku”:

* spēles gaitā, kad spēlētājam izbeidzas gājieni stundā, laiks paiet par stundu uz priekšu.
* kad tas pāriet jaunā dienā, mainas diena



4.7.att. Modulis “Mainīt Dienu, Laiku”

# 5. DATU STRUKTŪRAS APRAKSTS

Datu bāze sastāv no 4 tabulām, kas satur sevī informāciju par spēlētāju, ieročiem, pretiniekiem un līmeņiem.

1. Tabula “City” glabā datus par spēlētāju.
2. Tabula “Space” glabā datus par ieročiem.
3. Tabula “Player” glabā datus par pretiniekiem.

Tabula “City” ir saistīta ar tabulu “Space”

1. tabula

Tabulas “City” struktūra

| **Nr.** | **Nosaukums** | **Tips** | **Izmērs** | **Apraksts** |
| --- | --- | --- | --- | --- |
| 1. | City\_ID | int | 10 | Pilsētas unikālais kods |
| 2. | Items | array |  | Lietu masīvs |
| 3. | Resources | array |  | Resursu masīvs |
| 4. | Properties | array |  | Īpašumu masīvs |

Tabula “Space” ir saistīta ar tabulu “City” un “Player”

1. tabula

Tabulas “Space” struktūra

| **Nr.** | **Nosaukums** | **Tips** | **Izmērs** | **Apraksts** |
| --- | --- | --- | --- | --- |
| 1. | Space\_ID | int | 10 | Laukuma unikālais kods |
| 2. | Type | int | 5 | Laukuma tips |
| 3. | City\_ID | int | 10 | Pilsētas unikālā koda ārējā atslēga |
| 4. | Player\_ID | int | 10 | Spēlētāja unikālā koda ārējā atslēga |

Tabula “Player” ir saistīta ar tabulu “Space”

1. tabula

Tabulas “Player” struktūra

| **Nr.** | **Nosaukums** | **Tips** | **Izmērs** | **Apraksts** |
| --- | --- | --- | --- | --- |
| 1. | Player\_ID | int | 11 | Spēlētāja unikālais kods |
| 2. | Items | array |  | Lietu masīvs |
| 3. | Resources | array |  | Resursu masīvs |
| 4. | Properties | array |  | Īpašumu masīvs |
| 5. | Skills | array |  | Prasmju masīvs |
| 6. | Attributes | array |  | Atribūtu masīvs |

# 6. LIETOTĀJA CEĻVEDIS

## Sistēmas prasības aparatūrai un programmatūrai

Šī programma ir izstrādāta, lai darbotos uz mūsdienu datoriem, kas nozīmē, ka pat dažus gadus vecs dators būs piemērots. Svarīgākais ir tas, lai datoram būtu Windows operētājsistēma, jo programma veidota tieši šajai platformai. Windows 10 vai jaunāka versija būtu vispiemērotākā izvēle, lai nodrošinātu labāko veiktspēju un saderību.

Turklāt, lai gan šī programma nav resursu intensīva, ir vērts pievērst uzmanību dažiem aparatūras aspektiem. Pārliecinieties, ka jūsu datoram ir pietiekami daudz operatīvās atmiņas (RAM), vēlams vismaz 4 GB, lai nodrošinātu vienmērīgu darbību. Arī procesoram vajadzētu būt pietiekami jaudīgam, lai bez problēmām darbotos programma. Ja datoram ir SSD disks, tad tas var uzlabot programmas ielādes ātrumu.

Vēl viens būtisks faktors ir grafikas karte. Lai gan šī programma neprasa augstas veiktspējas grafikas karti, tā tomēr gūs labumu no jebkuras mūsdienu grafikas kartes. Ja jūsu datoram ir moderna grafikas karte, jūs varēsiet baudīt programmu ar maksimālajā vizuālajā kvalitātē.

Ir svarīgi arī pārliecināties, ka datoram ir pieejami jaunākie draiveri un operētājsistēmas atjauninājumi. Atjaunināti draiveri un operētājsistēma nodrošinās ne tikai labāku veiktspēju, bet arī augstāku drošības līmeni. Tas ir īpaši svarīgi, jo novecojuši draiveri vai operētājsistēma var radīt problēmas programmas darbībā vai pat izraisīt tās nesaderību. Ievērojot šos ieteikumus, dators spēs nodrošināt vienmērīgu un stabilu programmas darbību.

## Sistēmas instalācija un palaišana

Vispirms ir nepieciešams atvērt spēles mapi, kurā ir saglabāti visi spēles faili. Parasti šī mape atrodas vietā, kur esat instalējis spēli. To var atrast, atverot failu pārlūku un dodoties uz spēles instalācijas vietu. Ja neesat pārliecināts, kur atrodas spēles mape, to varat pārbaudīt spēles instalācijas laikā norādītajā vietā vai spēles saīsnes īpašībās, ja tāda ir izveidota uz darbvirsmas.

Kad esat atradis spēles mapi, nākamais solis ir atrast spēles “.EXE” failu. Šis fails ir galvenais izpildāmais fails, kas nepieciešams, lai sāktu spēli. Fails ir nosaukts pēc spēles nosaukuma vai tam līdzīgs. Šajā gadījumš tas ir nosaukts "TradeGame.exe". Rūpīgi pārskatiet visus failus un apakšmapes, lai atrastu pareizo izpildāmo failu.

Kad esat atradis “.EXE” failu, dubultklikšķiniet uz tā, lai palaistu spēli. Ja viss ir pareizi instalēts un nav nekādu problēmu ar failiem, spēlei būtu jāsāk darboties. Sekojot šīm darbībām, jūs veiksmīgi varēsiet palaist spēli un sākt to spēlēt.

## Programmas apraksts

Kad tiek atvērta spēle, ir redzams galvenās izvēlnes logs (skat. 6.1. att.), kurā ir 5 pogas, kad sāk spēli, bet, ja šis logs tiek atvērts spēles laikā pieejama arī sestā poga “Continue”.



6.1.att. Galvenās izvēlnes logs

Nospiežot “New Game” pogu tiek atvērts personāža izveides logs (skat. 6.2. att.), kurā vispirms spēlētājs var izvēlēties grūtības pakāpi, kurā vēlās spēlēt, ir norādīti resursi, kurus saņem katrā no grūtības pakāpēm.



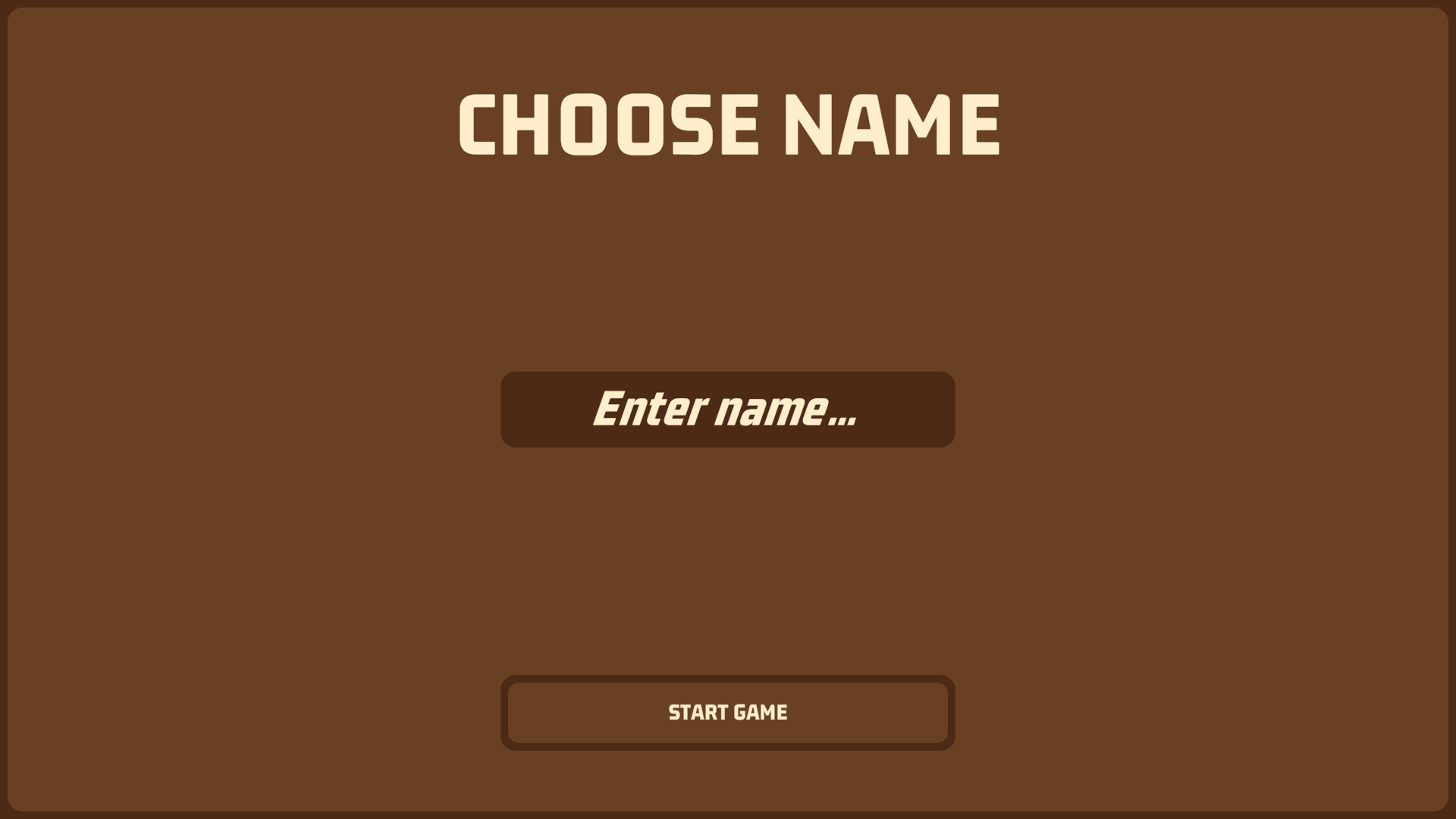
6.2.att. Personāža izveides loga grūtības pakāpes izvēles solis

Izvēloties kādu no grūtības pakāpēm tiek atvērts nākamais solis personāža izveides logā (skat. 6.3. att.), kurā var izvēlēties sava personāža prasmes un atribūtus. No izvēlētās grūtības pakāpes ir atkarīgs, ar cik prasmēm un atribūtu līmeņiem var sākt spēli.



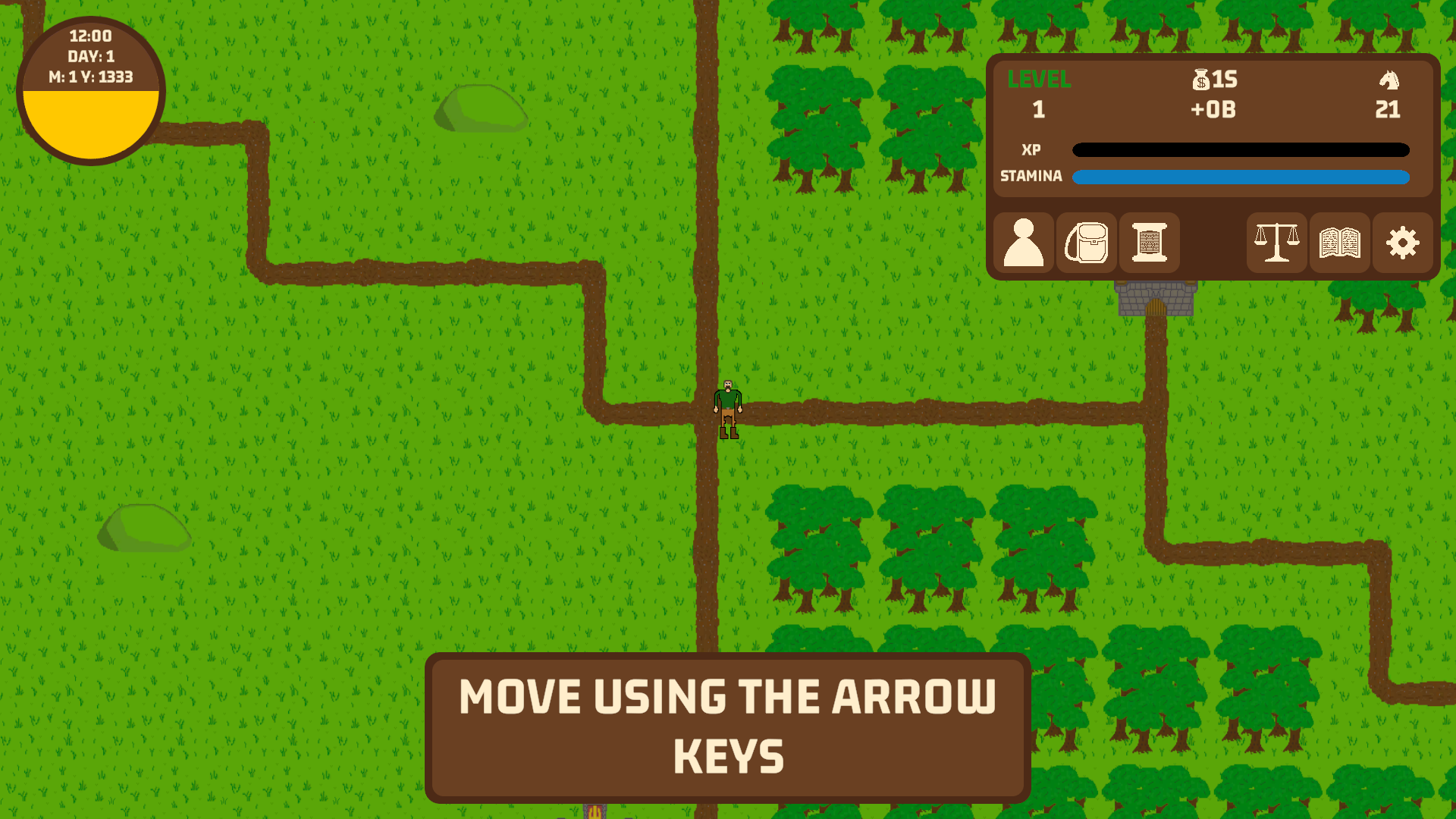
6.3.att. Personāža izveides loga prasmju un atribūtu izvēles solis

Kad apmierina izvēlētās prasmes un atribūti, nospiežot “Continue” pogu var turpināt uz pēdējo soli personāža izveides procesā, kurā var izvēlēties vārdu personāžam. Kad ir ievadīts vēlamais vārds, nospiežot “Start Game” pogu var sākas spēle un tiek atvērts kartes skats, kurā notiek kustība pasaulē.



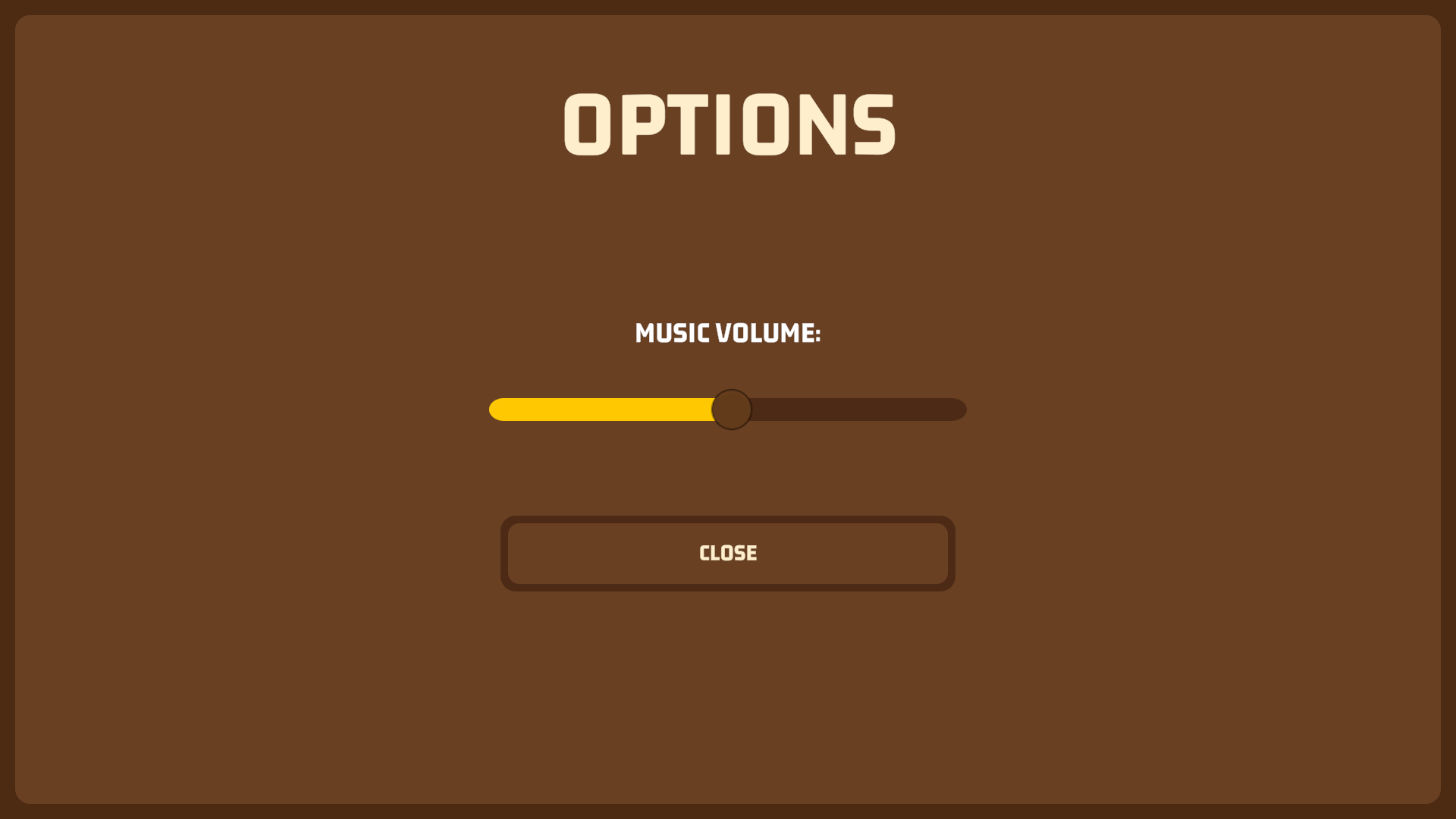
6.4.att. Personāža izveides loga personāža vārda izvēles solis

Kad apmierina izvēlētās prasmes un atribūti, nospiežot “Continue” pogu var turpināt uz pēdējo soli personāža izveides procesā, kurā var izvēlēties vārdu personāžam. Kad ir ievadīts vēlamais vārds, nospiežot “Start Game” pogu var sākas spēle un tiek atvērts kartes skats (skat. 6.5. att.), kurā notiek kustība pasaulē.



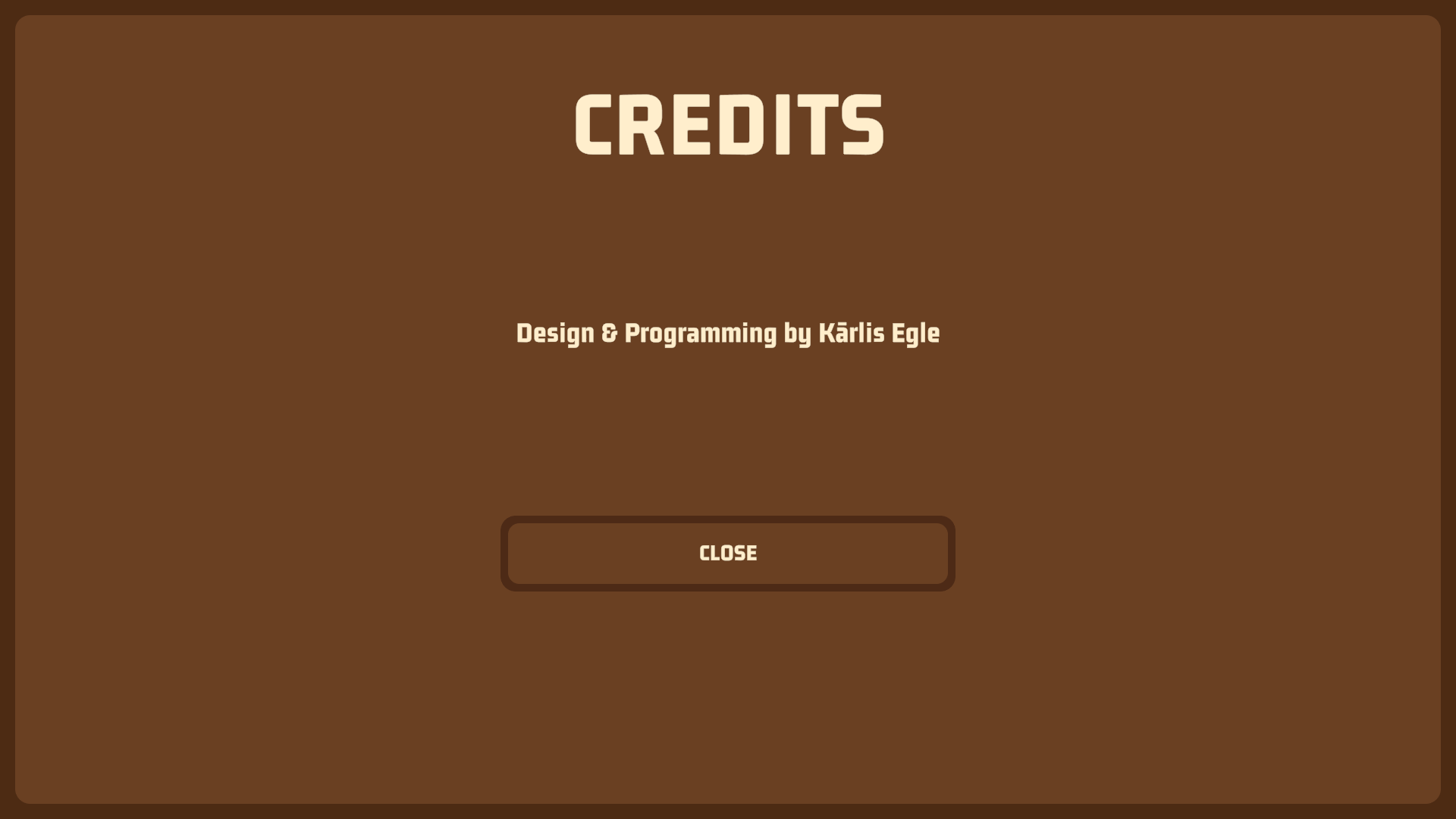
6.5.att. Kartes logs

Atgriežoties pie galvenās izvēlnes (skat. 6.1. att.), nospiežot “Continue” pogu spēlētājs tiek atgriezts uz kartes logu (skat 6.5. att.), nospiežot “Load Game” pogu tiek ielādēti dati no pēdējās reizes, kad spēlētājs saglabājās un tiek atvērts kartes logs (skat 6.5. att.), nospiežot “Options” pogu atveras opciju logs (skat. 6.6. att), kurā var mainīt mūzikas skaļumu vai atgriezties uz galveno izvēlni izmanotjot “Close” pogu.



6.6.att. Opciju logs

Tālāk galvenajā izvēlnē (skat. 6.1. att.), nospiežot “Credits” pogu tiek atvērts logs, kurā var redzēt izstrādātāju (skat 6.7. att.), un kurā var atgriezties uz galveno izvēlni izmanotjot “Close” pogu, pēdējā poga izvēlnē ir “Exit” poga, kas aizver spēli.



6.7.att. “Credits” logs

Kartes logā (skat. 6.5. att.) kreisajā augšējā pusē ir pulkstenis, kurā var redzēt atktūālo spēles laiku, dienu, mēnesi, gadu un saules rādītāju, loga labajā augšējā pusē ir informācīja par spēlētāju - līmenis, nauda, ieņēmumi, ātrums, pieredzes un izturības stabi - un zem tās ir sešas pogas, pirmā no tām atver personāža logu (skat. 6.8. att.), kurā var pārvaldīt prasmes, nospiežot uz izvēlētās prasmes, ja ir pieejami prasmju punkti, kurus var redzēt centra augšas kreisajā pusē, un atribūtu līmeņus, kurus var mainīt izmantojot “+” un “-” pogas pie attiecīgā atribūta, kas patērē atribūtu punktus, kuri norādīti centra augšas labajā pusē, kā arī var redzēt informāciju par personāžu. Šajā logā nospiežot “Close” pogu var atgriezties uz kartes logu.



6.8.att. Personāža logs

Nākamā kartes loga (skat. 6.5. att.) poga, atver personāža inventāra logu (skat. 6.9. att.), kurā var pārvaldīt visas lietas, kas pieder personāžam, kurus var filtrēt izmantojot “All items”, “Equipment”, “Food”, “Resources” un “Transport” pogas. Šajā logā nospiežot “Exit” pogu var atgriezties uz kartes logu.



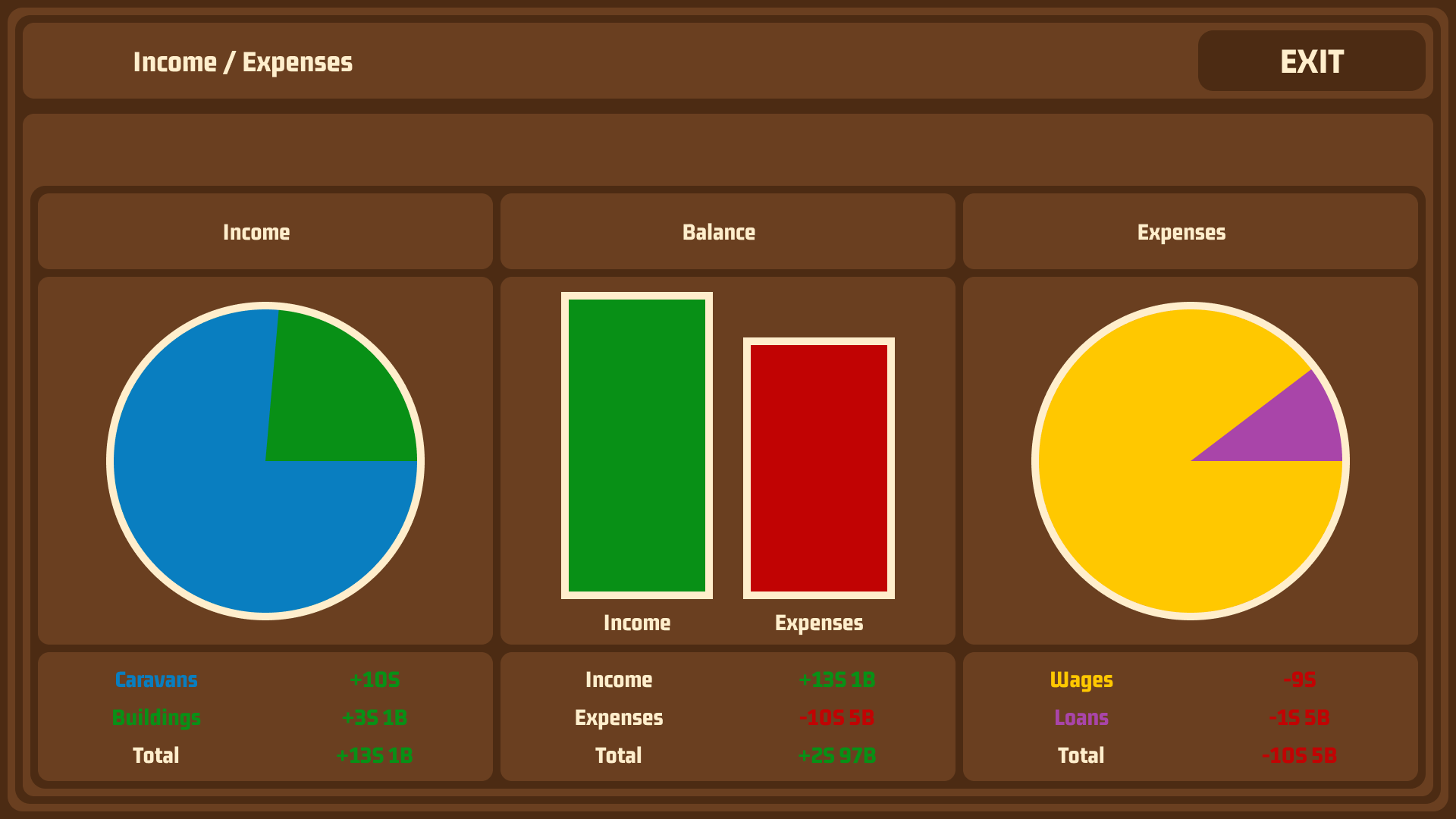
6.9.att. Inventāra logs

Trešā kartes loga (skat. 6.5. att.) poga, atver personāža uzdevumu logu (skat. 6.10. att.), kurā var pārvaldīt visus uzdevumus, kas ir personāžam, tos var filtrēt izmantojot “All Quests”, “Main”, “Side”, “Town”, “Active” un “Completed” pogas, lai tikai rādītu noteiktā tipa uzdevumus. Šajā logā nospiežot “Exit” pogu var atgriezties uz kartes logu.



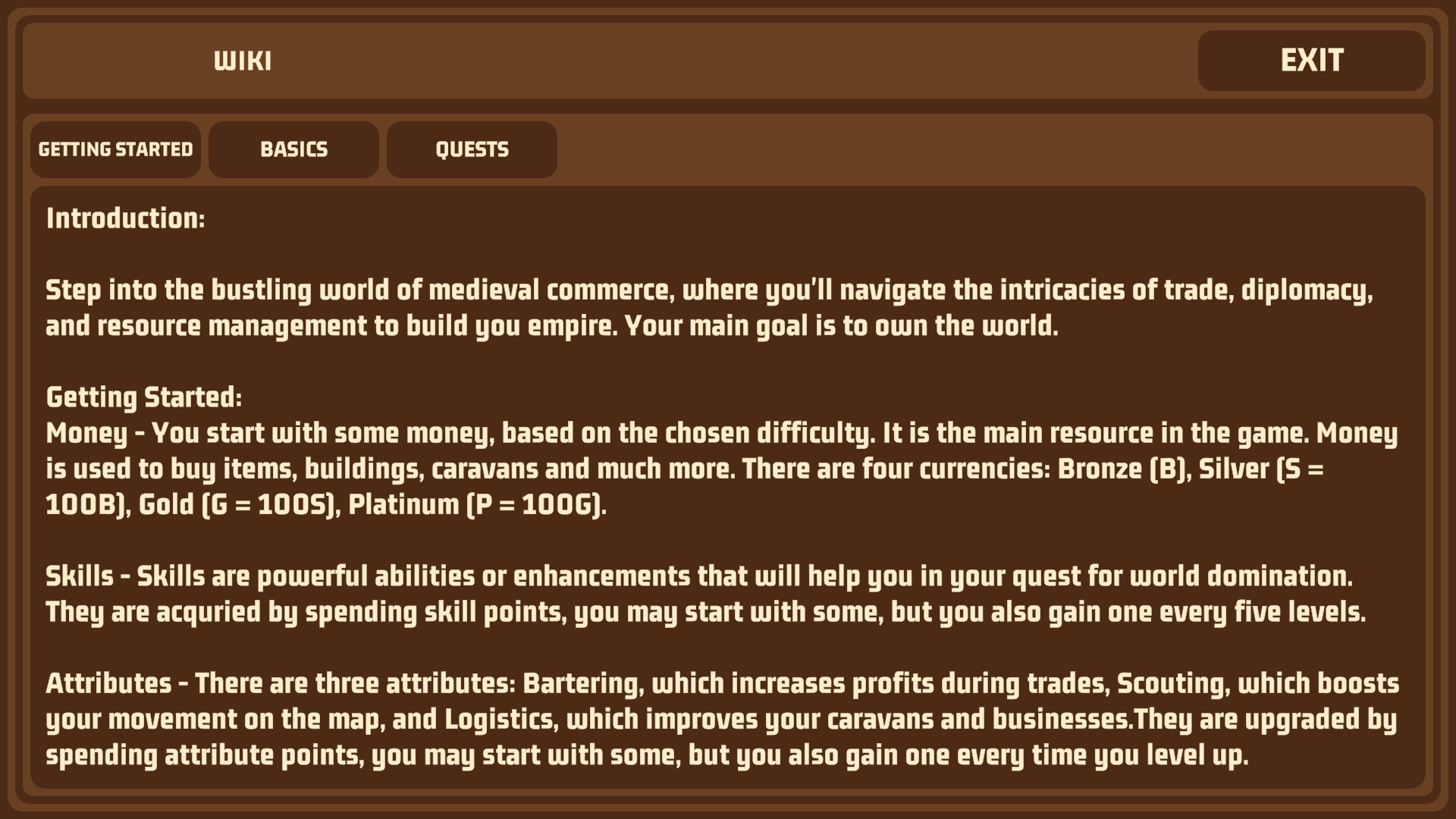
6.10.att. Uzdevumu logs

Nākama kartes loga (skat. 6.5. att.) poga, atver bilanses logu (skat. 6.11. att.), kurā var redzēt visus ienākumus un izmaksas. Šajā logā nospiežot “Exit” pogu var atgriezties uz kartes logu.



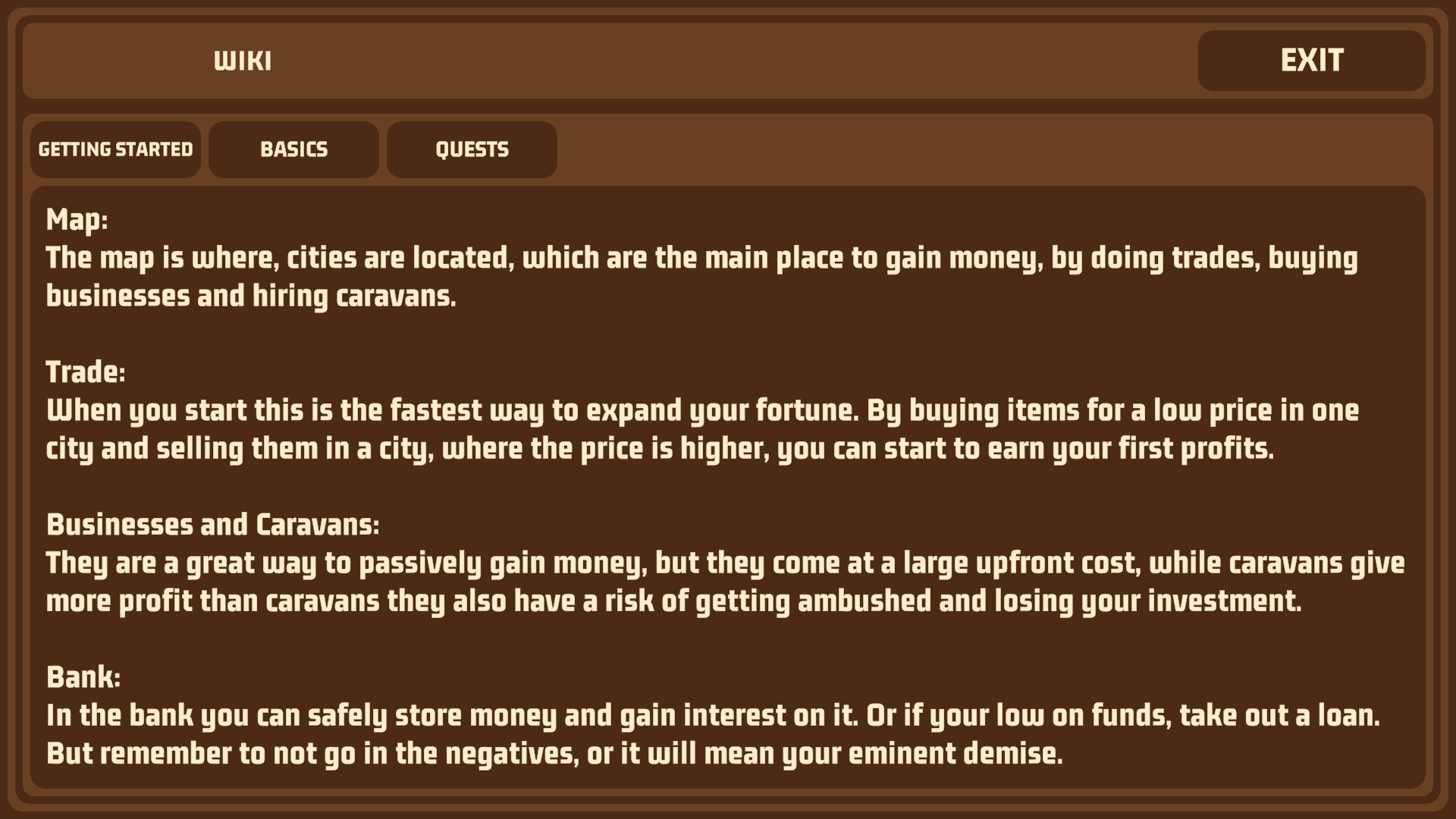
6.11.att. Bilances logs

Piektā kartes loga (skat. 6.5. att.) poga, atver spēles informācījas logu (skat. 6.12. att.), kurā var atrast informācīju par spēli. Šajā logā nospiežot “Exit” pogu var atgriezties uz kartes logu.



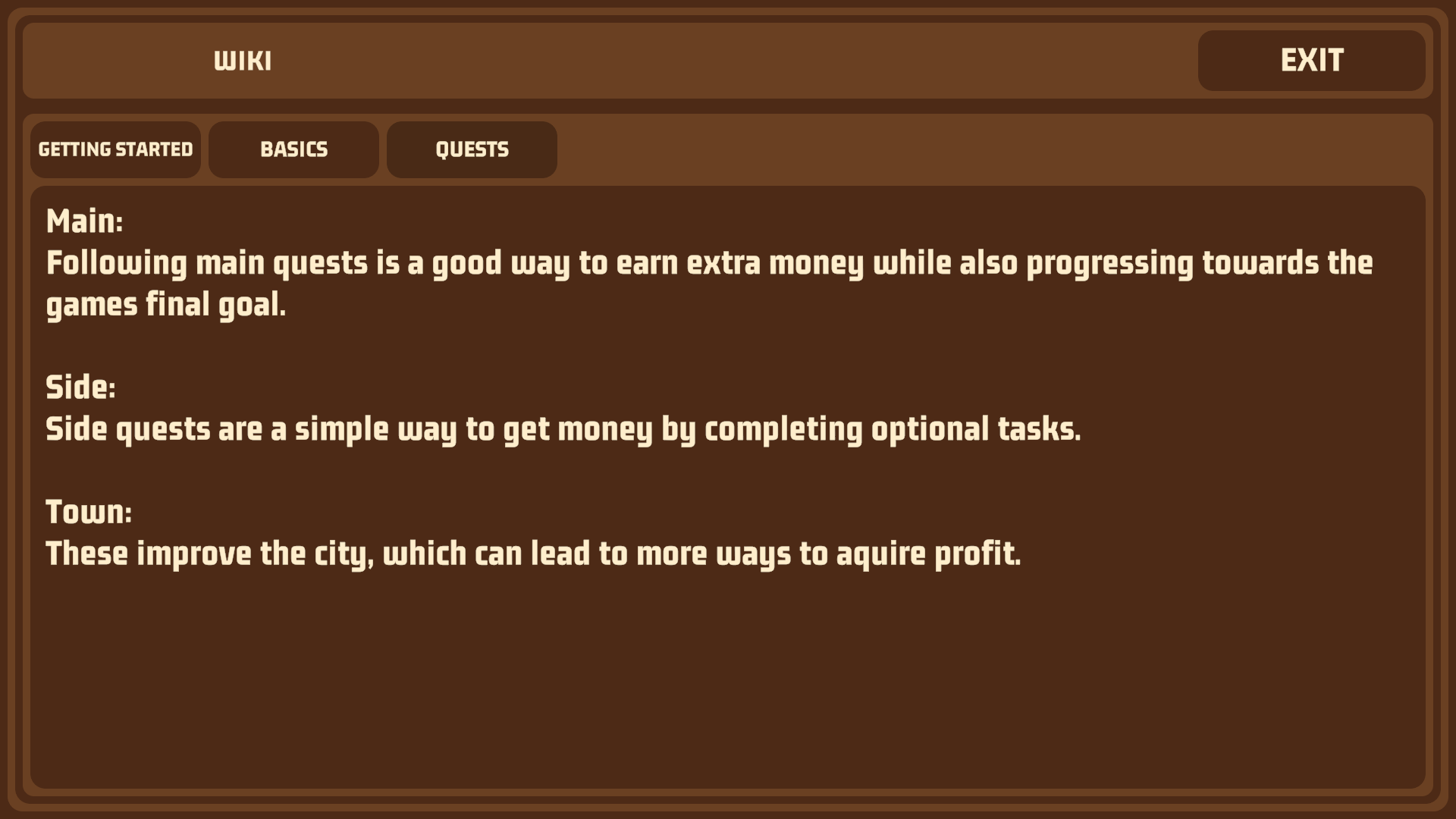
6.12.att. Informācījas loga “Getting Started” sadaļa

Nospiežot “Basics” pogu, atver spēles informācījas loga “Basics” sadaļu (skat. 6.13. att.), kurā ir informācīja par spēles pamata darbībām.



6.13.att. Informācījas loga “Basics” sadaļa

Nospiežot “Quests” pogu, atver spēles informācījas loga “Quests” sadaļu (skat. 6.14. att.), kurā ir informācīja par uzdevumiem.



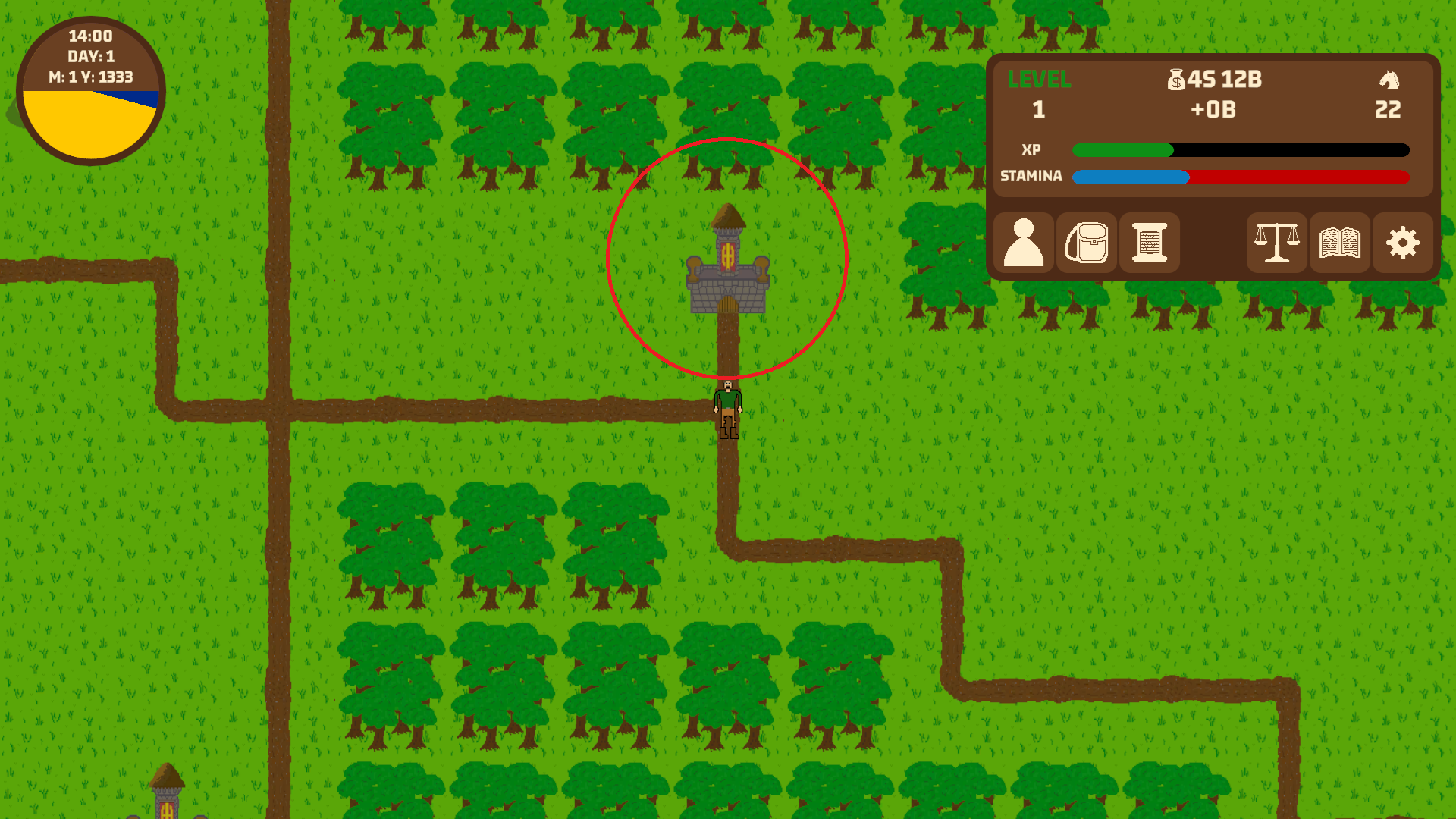
6.14.att. Informācījas loga “Quests” sadaļa

Pēdēja kartes loga (skat. 6.5. att.) poga, atver spēles kartes izvēlnes logu (skat. 6.15. att.). “Main Menu” poga atver galvenās izvēlnes logu (skat. 6.1. att.), “Load Game” un “Options” pogu funkcījas ir tādas pašas kā galvenajā izvēlnē, “Save Game” poga saglabā spēles progresu lokāli datorā, “Close” poga aizver kartes izvēlnes logu un spēlētājs nonāk atpakaļ kartes logā.



6.15.att. Kartes izvēlnes logs

Kartes logā (skat. 6.5. att.), kad ieiet pilsētas laukā (skat. 6.16. att.), tiek atvērts pilsētas logs (skat. 6.17. att.).



6.16.att. Kartes logs ar iezīmētu pilsētas laukumu

Pilsētas logs (skat. 6.17. att.) atveras “Stats” skatā, kurā var redzēt pilsētas nosaukumu, populācīju, bagātību un ēdiena rezerves, kā arī šo statistiku pārmaiņas pēdējās piecās dienās, loga centra augšā redzams spēlētāja naudas daudzums. Izmantojot “Exit” pogu var iziet no pilsētas un atgriezties kartes logā.



6.17.att. Pilsētas loga “Stats” sadaļa

Pilsētas loga (skat. 6.17. att.) “Trade” poga atver “Trade” sadaļas “Buy” skatu (skat. 6.18. att.), kurā var iegādāties lietas no pilsētas inventāra saraksta. Katrai lietai sarakstā ir norādīts vārds, apraksts, daudzums pilsētā, vietējā cena un darbības, kuras ar to var veikt, kas šajā skatā ir tikai pirkt noteikto lietu. Šajā sadaļā ir arī “Buy” un “Sell” pogas, kas atver attiecīgos skatus.



6.18.att. Pilsētas loga “Trade” sadaļas “Buy” skats

“Sell” poga atver “Trade” sadaļas “Sell” skatu (skat. 6.19. att.), kurā var pārdot lietas no personāža inventāra saraksta pilsētai. Katrai lietai sarakstā ir norādīts vārds, apraksts, daudzums pilsētā, vietējā cena un darbības, kuras ar to var veikt, kas šajā skatā ir tikai pārdot noteikto lietu.



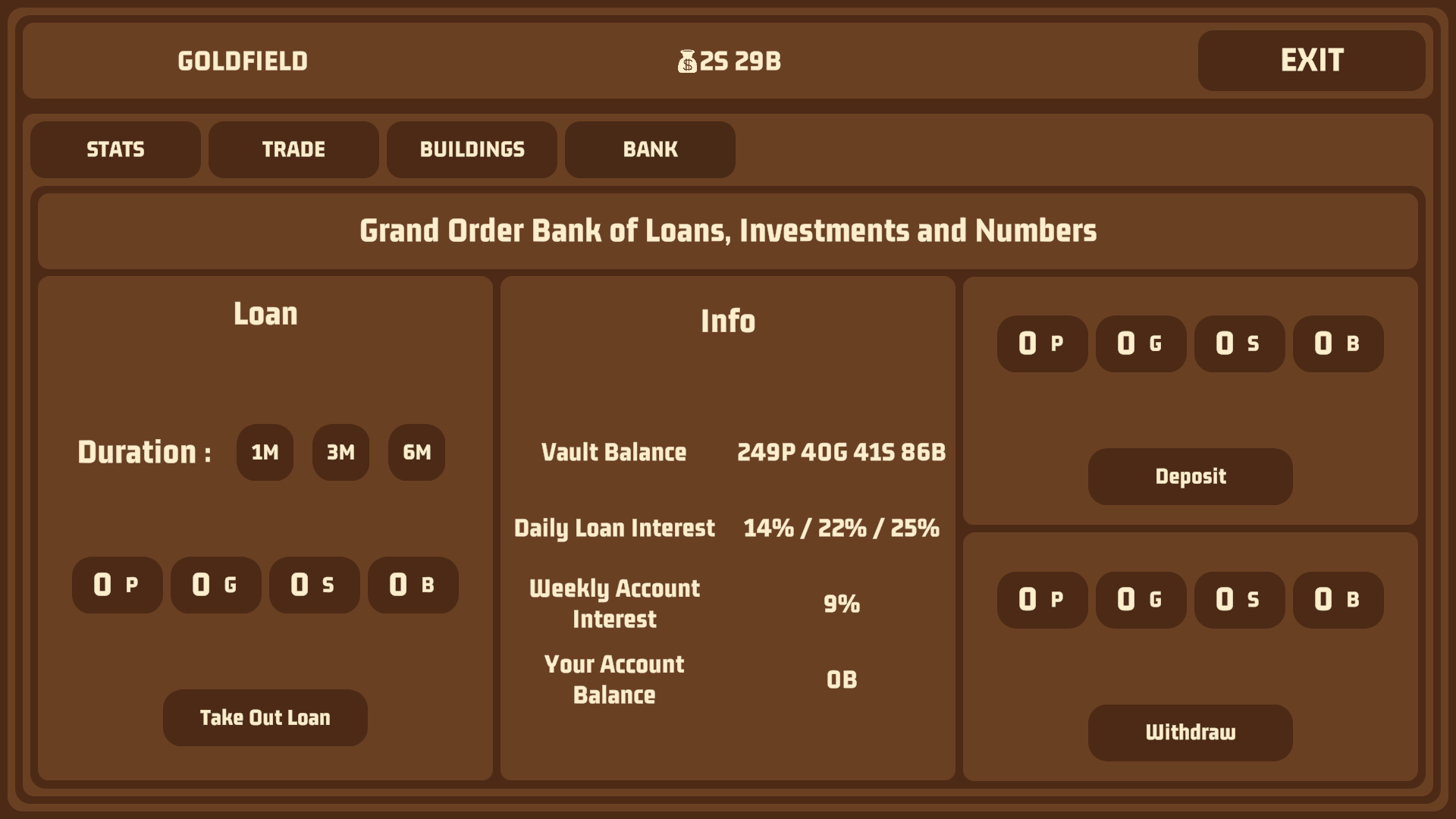
6.19.att. Pilsētas loga “Trade” sadaļas “Sell” skats

Pilsētas loga (skat. 6.17. att.) “Buildings” poga atver “Buildings” sadaļu (skat. 6.20. att.), kurā var pirkt vai pārdot pilsētas īpašumus. Katram īpašumam sarakstā ir norādīts vārds, apraksts, ienākumi, cena un darbības, kuras ar to var veikt, kas var būt pārdot vai pirkt noteikto īpašumu.



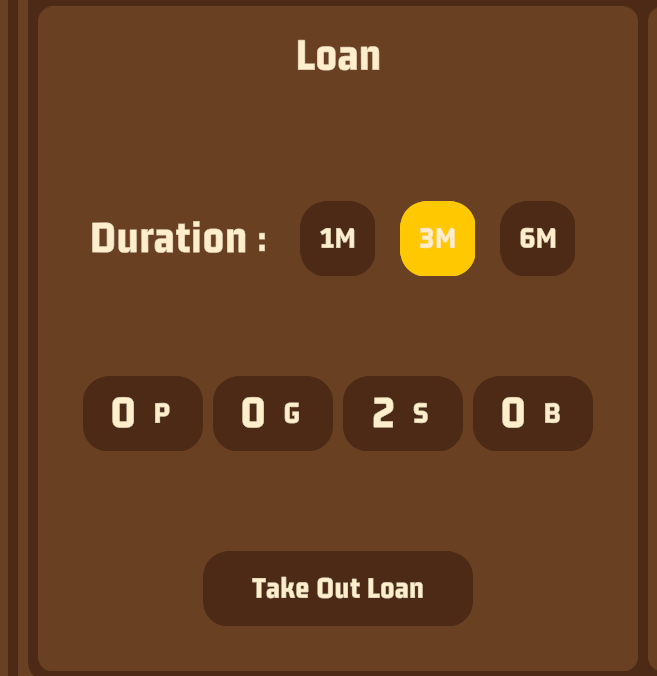
6.20.att. Pilsētas loga “Buildings” sadaļa

Pilsētas loga (skat. 6.17. att.) “Bank” poga atver “Bank” sadaļu (skat. 6.21. att.), kurā var redzēt informācīju par pilsētas banku un pārvaldīt savu kontu tajā.



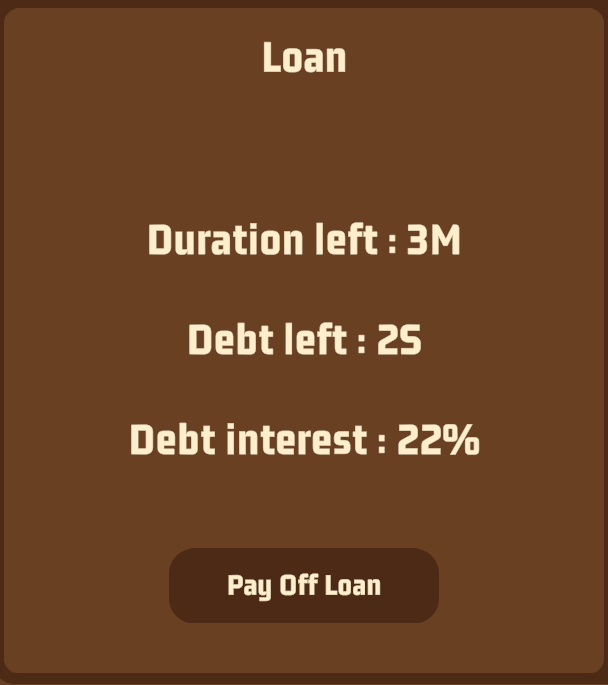
6.21.att. Pilsētas loga “Bank” sadaļa

“Bank” sadaļā (skat. 6.21. att.) iespējams izņemt aizņēmumu (skat. 6.22. att.).



6.22.att. Pilsētas loga “Bank” sadaļas aizpildīta aizņēmuma sadaļa

“Bank” sadaļā (skat. 6.21. att.) iespējams arī to atmaksāt (skat. 6.23. att.).



6.23.att. Pilsētas loga “Bank” sadaļas aizņēmuma atmaksas sadaļa

“Bank” sadaļā (skat. 6.21. att.) arī var ielikt un izņemt naudu no sava konta ievadot vēlamo daudzumu attiecīgajā sadaļā un nospiežot attiecīgo pogu.

## 6.4. Testa Piemērs

| **Nr.** | **Prasības nr.** | **Prasības nosaukums** | **Ievaddati / situācijas apraksts** | **Sagaidāmais rezultāts** | **Statuss** |
| --- | --- | --- | --- | --- | --- |
| 1. | 1. | Jānodrošina personāža kustība | Nospiests labais taustiņš | Personāžs kustās pa labi kartē | Pareizi |
| 2. | Nospiests kreisais taustiņš | Personāžs kustās pa kreisi kartē | Pareizi |
| 3. | 3. | Jānodrošina īpašumu, lietu, prasmju un atribūtu parvaldīšanu. | Nopērk lietu no pilsētas | Tā tiek pārvietota no pilsētas inventāra uz spēlētāja | Pareizi |
| 4. | Pievieno atribūtu līmeni | Tas paaugstinās un tiek patērēts 1 atribūtu punkts | Pareizi |
| 5. | 4. | Jānodrošina spēja atvērt galveno izvēlni. | Atver spēli | Tā atveras | Pareizi |
| 6. | Kartes izvēlnē nospiež  “Main Menu” pogu | Tā atveras | Pareizi |

# 

# NOBEIGUMS

Izstrādājot šo kvalifikācijas darbu, esmu izveidojis stratēģijas datorspēli, kas piedāvā autentisku viduslaiku ekonomikas un tirdzniecības pieredzi. Spēles veidošanas procesā tika ņemtas vērā vēsturiskās realitātes, lai spēlētāji varētu iegūt pēc iespējas precīzāku priekšstatu par viduslaiku tirgotāju un amatnieku dzīvi un izaicinājumiem.

Spēles izstrādes laikā tika pielietotas dažādas programmēšanas un dizaina prasmes, kas ļāva radīt produktu, kas ir gan vizuāli pievilcīgs, gan arī funkcionāli bagāts. Spēlētājiem tiek piedāvāta iespēja pieņemt stratēģiskus lēmumus, pārvaldīt resursus un mijiedarboties ar dažādiem spēles elementiem, kas padara spēli aizraujošu un izaicinošu.

Projekta rezultātā esmu guvis dziļāku izpratni par spēļu dizainu un izstrādi. Ceru, ka šī spēle ne tikai izklaidēs stratēģijas spēļu cienītājus, bet arī sniegs vērtīgu izglītojošu pieredzi par ekonomiku.

Kopumā šis projekts ir bijis liels solis uz priekšu manā profesionālajā izaugsmē, sniedzot man iespēju attīstīt savas prasmes un radīt kaut ko, kas atbilst maniem standartiem un spēj ilglaicīgi piesaistīt spēlētāju interesi.

# 

# INFORMĀCIJAS AVOTI

* [ANG] Unity 3D dokumentācija - <https://docs.unity3d.com/Manual/index.html>.
* [ANG] Unity Forums - <https://forum.unity.com>.
* [ANG] Youtube - <https://youtube.com>.
* [ANG] W3Schools - <https://www.w3schools.com/cs/index.php>
* [ANG] Stack Overflow - <https://stackoverflow.com>
* [ANG] Udemy - <https://www.udemy.com>
* [ANG] GeeksForGeeks - <https://www.geeksforgeeks.org/csharp-programming-language>
* [ANG] Code Academy - <https://www.codecademy.com/learn/learn-c-sharp>
* [ANG] Unity dokumentācīja - <https://docs.unity.com/>

# 

# PIELIKUMI

**1. pielikums**

**Programmas pirmteksts**

**AudioSlider.cs**

using UnityEngine;

using UnityEngine.UI;

public class AudioSlider : MonoBehaviour

{

public Slider slider;

void Start()

{

if (PlayerPrefs.HasKey(MainManager.Instance.VolumePrefsKey))

{

MainManager.Instance.audioSource.volume = PlayerPrefs.GetFloat(MainManager.Instance.VolumePrefsKey);

}

slider.value = MainManager.Instance.audioSource.volume;

// Add a listener to respond to changes in the slider value

slider.onValueChanged.AddListener(delegate { OnSliderChanged(); });

}

// Update the audio volume when the slider value changes

void OnSliderChanged()

{

MainManager.Instance.audioSource.volume = slider.value;

PlayerPrefs.SetFloat(MainManager.Instance.VolumePrefsKey, MainManager.Instance.audioSource.volume);

PlayerPrefs.Save();

}

}

**CameraFollow.cs**

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class CameraFollow : MonoBehaviour

{

public Transform player;

public Vector3 offset;

void Update()

{

transform.position = new Vector3(player.position.x + offset.x, player.position.y + offset.y, offset.z); // Camera follows the player with specified offset position

}

}

**ChracterCreation.cs**

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

using UnityEngine.UI;

using TMPro;

using UnityEngine.SceneManagement;

public class CharacterCreation : MonoBehaviour

{

public TextMeshProUGUI AttrText;

public TextMeshProUGUI SkillText;

public GameObject AttribuesBlock;

public GameObject SkillsBlock;

public List<TMP\_Text> AttributeLevels;

public GameObject[,] SkillsArray = new GameObject[3, 5];

public TextMeshProUGUI NameText;

public void SelectDifficulty(int diff)

{

// Set Difficulty

PlayerManager.Instance.Difficulty = diff;

PlayerManager.Instance.InitSkillsAndAttr();

// Prepare Skills And Attributes View

AttrText.text = "<sprite name=\"Character Icon\"> " + PlayerManager.Instance.AttrPoints.ToString();

SkillText.text = "<sprite name=\"Quest Icon\"> " + PlayerManager.Instance.SkillPoints.ToString();

foreach (TMP\_Text tmpText in AttribuesBlock.GetComponentsInChildren<TMP\_Text>())

{

if (tmpText.gameObject.name == "Level\_Txt")

{

AttributeLevels.Add(tmpText);

}

}

foreach (Transform childTransform in AttribuesBlock.transform){

foreach (Transform grandchildTransform in childTransform){

if (grandchildTransform.gameObject.name == "Level\_Txt"){

TMP\_Text[] tmpTexts = grandchildTransform.gameObject.GetComponentsInChildren<TMP\_Text>();

foreach (TMP\_Text tmpText in tmpTexts){

AttributeLevels.Add(tmpText);

}

}

}

}

int index = 0;

foreach (TMP\_Text levelText in AttributeLevels){

levelText.text = "<color=#089016>LEVEL</color> " + PlayerManager.Instance.Attributes[index].ToString();

index++;

}

int numRows = SkillsBlock.transform.childCount;

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

{

Transform childTransform = SkillsBlock.transform.GetChild(i);

int numCols = 5;

for (int j = 0; j < numCols; j++)

{

Transform grandchildTransform = childTransform.GetChild(j);

SkillsArray[i, j] = grandchildTransform.gameObject;

}

}

for (int row = 0; row < numRows; row++)

{

int numCols = SkillsArray.GetLength(1);

for (int col = 0; col < numCols; col++)

{

GameObject element = SkillsArray[row, col];

Image image = element.transform.Find("Skill Description BG")?.gameObject.GetComponent<Image>();

if (PlayerManager.Instance.Skills[row, col])

{

image.color = new Color32(8, 144, 22, 255);

}

else

{

image.color = new Color32(106, 63, 32, 255);

}

}

}

}

public void AddAttr(int attribute)

{

ChangeAttribute(attribute, 1);

}

public void SubtractAttr(int attribute)

{

ChangeAttribute(attribute, -1);

}

public void ChangeAttribute(int attribute, int change)

{

if (PlayerManager.Instance.Attributes[attribute] + change > 0 && change <= PlayerManager.Instance.AttrPoints)

{

PlayerManager.Instance.Attributes[attribute] += change;

PlayerManager.Instance.AttrPoints -= change;

AttributeLevels[attribute].text = "<color=#089016>LEVEL</color> " + PlayerManager.Instance.Attributes[attribute].ToString();

AttrText.text = "<sprite name=\"Character Icon\"> " + PlayerManager.Instance.AttrPoints.ToString();

}

}

public void ToggleSkill(int skill)

{

int set = 0;

while (skill > 4)

{

skill -= 5;

set++;

}

if (PlayerManager.Instance.Skills[set, skill])

{

PlayerManager.Instance.Skills[set, skill] = false;

PlayerManager.Instance.SkillPoints += 1;

Image image = SkillsArray[set, skill].transform.Find("Skill Description BG")?.gameObject.GetComponent<Image>();

image.color = new Color32(106, 63, 32, 255);

SkillText.text = "<sprite name=\"Quest Icon\"> " + PlayerManager.Instance.SkillPoints.ToString();

}

else if (PlayerManager.Instance.SkillPoints > 0)

{

PlayerManager.Instance.Skills[set, skill] = true;

PlayerManager.Instance.SkillPoints -= 1;

Image image = SkillsArray[set, skill].transform.Find("Skill Description BG")?.gameObject.GetComponent<Image>();

image.color = new Color32(8, 144, 22, 255);

SkillText.text = "<sprite name=\"Quest Icon\"> " + PlayerManager.Instance.SkillPoints.ToString();

}

}

public void StartGame()

{

if(PlayerManager.Instance.Name != "")

{

PlayerManager.Instance.MaxStamina = 29 + PlayerManager.Instance.Attributes[2];

PlayerManager.Instance.Stamina = (float)PlayerManager.Instance.MaxStamina;

SceneManager.LoadSceneAsync("MapView");

PlayerManager.Instance.LoadPlayerPosition();

}

}

public void ChooseName()

{

PlayerManager.Instance.Name = NameText.text;

}

void Update()

{

}

}

**CharacterMenu.cs**

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

using UnityEngine.UI;

using TMPro;

public class CharacterMenu : MonoBehaviour

{

public TextMeshProUGUI NameAndLevelText;

public TextMeshProUGUI AttrText;

public TextMeshProUGUI SkillText;

public GameObject AttribuesBlock;

public GameObject SkillsBlock;

public List<TMP\_Text> AttributeLevels;

public GameObject[,] SkillsArray = new GameObject[3, 5];

public Slider XPBar;

public TextMeshProUGUI XPText;

void Update()

{

NameAndLevelText.text = PlayerManager.Instance.Name + "\n";

NameAndLevelText.text += "<color=#089016>LEVEL</color> " + PlayerManager.Instance.PlayerLevel.ToString();

XPBar.value = ((float)PlayerManager.Instance.XP / (float)PlayerManager.Instance.GetXpToNextLevel()) \* 100;

XPText.text = "XP\n" + PlayerManager.Instance.XP.ToString() + "/" + PlayerManager.Instance.GetXpToNextLevel().ToString();

AttrText.text = "<sprite name=\"Character Icon\"> " + PlayerManager.Instance.AttrPoints.ToString();

SkillText.text = "<sprite name=\"Quest Icon\"> " + PlayerManager.Instance.SkillPoints.ToString();

int index = 0;

foreach (TMP\_Text levelText in AttributeLevels)

{

levelText.text = "<color=#089016>LEVEL</color> " + PlayerManager.Instance.Attributes[index].ToString();

index++;

}

int numRows = SkillsBlock.transform.childCount;

for (int row = 0; row < numRows; row++)

{

int numCols = SkillsArray.GetLength(1);

for (int col = 0; col < numCols; col++)

{

GameObject element = SkillsArray[row, col];

Image image = element.transform.Find("Skill Description BG")?.gameObject.GetComponent<Image>();

if (PlayerManager.Instance.Skills[row, col])

{

image.color = new Color32(8, 144, 22, 255);

}

else

{

image.color = new Color32(106, 63, 32, 255);

}

}

}

}

public void AddAttr(int attribute)

{

ChangeAttribute(attribute, 1);

}

public void SubtractAttr(int attribute)

{

ChangeAttribute(attribute, -1);

}

public void ChangeAttribute(int attribute, int change)

{

if (PlayerManager.Instance.Attributes[attribute] + change > 0 && change <= PlayerManager.Instance.AttrPoints)

{

PlayerManager.Instance.Attributes[attribute] += change;

PlayerManager.Instance.AttrPoints -= change;

AttributeLevels[attribute].text = "<color=#089016>LEVEL</color> " + PlayerManager.Instance.Attributes[attribute].ToString();

AttrText.text = "<sprite name=\"Character Icon\"> " + PlayerManager.Instance.AttrPoints.ToString();

}

}

public void ToggleSkill(int skill)

{

int set = 0;

while (skill > 4)

{

skill -= 5;

set++;

}

if (PlayerManager.Instance.Skills[set, skill])

{

PlayerManager.Instance.Skills[set, skill] = false;

PlayerManager.Instance.SkillPoints += 1;

Image image = SkillsArray[set, skill].transform.Find("Skill Description BG")?.gameObject.GetComponent<Image>();

image.color = new Color32(106, 63, 32, 255);

SkillText.text = "<sprite name=\"Quest Icon\"> " + PlayerManager.Instance.SkillPoints.ToString();

}

else if (PlayerManager.Instance.SkillPoints > 0)

{

PlayerManager.Instance.Skills[set, skill] = true;

PlayerManager.Instance.SkillPoints -= 1;

Image image = SkillsArray[set, skill].transform.Find("Skill Description BG")?.gameObject.GetComponent<Image>();

image.color = new Color32(8, 144, 22, 255);

SkillText.text = "<sprite name=\"Quest Icon\"> " + PlayerManager.Instance.SkillPoints.ToString();

}

}

void Start()

{

foreach (TMP\_Text tmpText in AttribuesBlock.GetComponentsInChildren<TMP\_Text>())

{

if (tmpText.gameObject.name == "Level\_Txt")

{

AttributeLevels.Add(tmpText);

}

}

foreach (Transform childTransform in AttribuesBlock.transform)

{

foreach (Transform grandchildTransform in childTransform)

{

if (grandchildTransform.gameObject.name == "Level\_Txt")

{

TMP\_Text[] tmpTexts = grandchildTransform.gameObject.GetComponentsInChildren<TMP\_Text>();

foreach (TMP\_Text tmpText in tmpTexts)

{

AttributeLevels.Add(tmpText);

}

}

}

}

int numRows = SkillsBlock.transform.childCount;

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

{

Transform childTransform = SkillsBlock.transform.GetChild(i);

int numCols = 5;

for (int j = 0; j < numCols; j++)

{

Transform grandchildTransform = childTransform.GetChild(j);

SkillsArray[i, j] = grandchildTransform.gameObject;

}

}

}

}

**CityMenu.cs**

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

using UnityEngine.UI;

using System.Linq;

using TMPro;

public class CityMenu : MonoBehaviour

{

[Header("Stats Menu")]

public GameObject StatsTab;

// Graphs

public RectTransform graphPanel0;

public RectTransform graphPanel1;

public RectTransform graphPanel2;

// Texts

public TextMeshProUGUI CityName;

public TextMeshProUGUI PopTxt;

public TextMeshProUGUI WealthTxt;

public TextMeshProUGUI FoodTxt;

public TextMeshProUGUI PlayerMoney;

// Graph settings

public float dotScale = 25f;

public Sprite circleSprite;

public float margin = 30f;

public float lineWidth = 20f;

public Color positiveChangeColor = Color.green;

public Color negativeChangeColor = Color.red;

[Header("Trade Menu")]

public GameObject TradeTab;

public GameObject BuySellBtns;

public GameObject buyListRowPrefab;

public GameObject sellListRowPrefab;

public RectTransform tradeContentPanel;

[Header("Buildings Menu")]

public GameObject BuildingsTab;

public GameObject buildingListRowPrefab;

public RectTransform buildingsContentPanel;

[Header("Bank Menu")]

public GameObject BankTab;

// Loan

public GameObject LoanTake;

public MoneyInput LoanMoneyInput;

public GameObject LoanPay;

// Info

public TextMeshProUGUI ValutBalanceText;

public TextMeshProUGUI LoanInterestText;

public TextMeshProUGUI AccountInterestText;

public TextMeshProUGUI AccountBalanceText;

// Deposit

public MoneyInput DepositMoneyInput;

// Withdraw

public MoneyInput WithdrawMoneyInput;

// Durration

public GameObject Duration1;

public GameObject Duration3;

public GameObject Duration6;

[Header("Caravans Menu")]

public GameObject CaravansTab;

// General

private int currentTileID;

private float buyModifier = 1.25f;

private int duration;

public void UpdateCityMenu(int tileID)

{

currentTileID = tileID;

StatsTab.SetActive(true);

TradeTab.SetActive(false);

BuildingsTab.SetActive(false);

BankTab.SetActive(false);

CaravansTab.SetActive(false);

// Draw Stats

UpdateStats();

// Draw Trade

UpdateList(0);

LayoutRebuilder.ForceRebuildLayoutImmediate(tradeContentPanel);

float contentHeight = tradeContentPanel.rect.height;

tradeContentPanel.anchoredPosition = new Vector2(tradeContentPanel.anchoredPosition.x, -(contentHeight / 2));

// Draw Bank

UpdateBank();

}

public void UpdateStats()

{

TileInfo tileInfo = MainManager.Instance.TileInfoMap[currentTileID];

float[] TilePop = tileInfo.TilePop.Select(x => (float)x).ToArray();

float[] TileWealth = tileInfo.TileWealth.Select(x => (float)x).ToArray();

float[] TileFood = tileInfo.TileFood.Select(x => (float)x).ToArray();

CityName.text = tileInfo.TileName;

PlayerMoney.text = "<sprite name=\"Money Icon\">" + MainManager.Instance.MoneyToString(PlayerManager.Instance.Money);

PopTxt.text = tileInfo.TilePop[5].ToString();

WealthTxt.text = MainManager.Instance.MoneyToString((long)tileInfo.TileWealth[5]);

FoodTxt.text = tileInfo.TileFood[5].ToString() + " Days";

MainManager.Instance.PlotGraph(graphPanel0, TilePop, dotScale, circleSprite, margin, lineWidth, positiveChangeColor, negativeChangeColor);

MainManager.Instance.PlotGraph(graphPanel1, TileWealth, dotScale, circleSprite, margin, lineWidth, positiveChangeColor, negativeChangeColor);

MainManager.Instance.PlotGraph(graphPanel2, TileFood, dotScale, circleSprite, margin, lineWidth, positiveChangeColor, negativeChangeColor);

}

public void UpdateBank()

{

Bank bank = MainManager.Instance.TileInfoMap[currentTileID].Bank;

// Output Info

ValutBalanceText.text = MainManager.Instance.MoneyToString(bank.VaultBalance);

LoanInterestText.text = "";

foreach (var interest in bank.LoanInterest)

{

LoanInterestText.text += interest + "% / ";

}

LoanInterestText.text = LoanInterestText.text.Remove(LoanInterestText.text.Length - 2);

int finalAccInt = bank.AccountInterest;

if (PlayerManager.Instance.Skills[0, 4])

{

finalAccInt += 5;

}

AccountInterestText.text = finalAccInt + "%";

AccountBalanceText.text = MainManager.Instance.MoneyToString(bank.PlayerBalance);

PlayerMoney.text = "<sprite name=\"Money Icon\">" + MainManager.Instance.MoneyToString(PlayerManager.Instance.Money);

// Loans

if(PlayerManager.Instance.Loans.Any(loan => loan.BankTileID == currentTileID))

{

LoanPay.SetActive(true);

LoanTake.SetActive(false);

Loan bankLoan = PlayerManager.Instance.Loans.FirstOrDefault(loan => loan.BankTileID == currentTileID);

TextMeshProUGUI[] payTexts = LoanPay.transform.GetComponentsInChildren<TextMeshProUGUI>();

payTexts[1].text = "Duration left : " + bankLoan.DurationLeft.TimeToString();

payTexts[2].text = "Debt left : " + MainManager.Instance.MoneyToString(bankLoan.DebtLeft);

payTexts[3].text = "Debt interest : " + bankLoan.Interest +"%";

}

else

{

LoanPay.SetActive(false);

LoanTake.SetActive(true);

}

}

public void SelectDuration(int selectedDuration)

{

duration = selectedDuration;

Image image1 = Duration1.GetComponent<Image>();

Image image3 = Duration3.GetComponent<Image>();

Image image6 = Duration6.GetComponent<Image>();

switch (duration)

{

case 1:

image1.color = new Color32(255, 200, 0, 255);

image3.color = new Color32(77, 43, 19, 255);

image6.color = new Color32(77, 43, 19, 255);

break;

case 3:

image1.color = new Color32(77, 43, 19, 255);

image3.color = new Color32(255, 200, 0, 255);

image6.color = new Color32(77, 43, 19, 255);

break;

case 6:

image1.color = new Color32(77, 43, 19, 255);

image3.color = new Color32(77, 43, 19, 255);

image6.color = new Color32(255, 200, 0, 255);

break;

default:

image1.color = new Color32(77, 43, 19, 255);

image3.color = new Color32(77, 43, 19, 255);

image6.color = new Color32(77, 43, 19, 255);

break;

}

}

public void BankDeposit()

{

int deposit = DepositMoneyInput.GetInputMoney();

if ( PlayerManager.Instance.Money >= deposit)

{

if(!PlayerManager.Instance.BanksWithFunds.Any(bankWF => bankWF.TileID == currentTileID))

{

PlayerManager.Instance.BanksWithFunds.Add(MainManager.Instance.TileInfoMap[currentTileID].Bank);

}

PlayerManager.Instance.Money -= deposit;

MainManager.Instance.TileInfoMap[currentTileID].Bank.VaultBalance += deposit;

MainManager.Instance.TileInfoMap[currentTileID].Bank.PlayerBalance += deposit;

UpdateBank();

DepositMoneyInput.ClearInputs();

}

}

public void BankWithdraw()

{

int withdraw = WithdrawMoneyInput.GetInputMoney();

Bank bank = MainManager.Instance.TileInfoMap[currentTileID].Bank;

if (bank.VaultBalance >= withdraw && bank.PlayerBalance >= withdraw)

{

PlayerManager.Instance.Money += withdraw;

MainManager.Instance.TileInfoMap[currentTileID].Bank.VaultBalance -= withdraw;

MainManager.Instance.TileInfoMap[currentTileID].Bank.PlayerBalance -= withdraw;

if(MainManager.Instance.TileInfoMap[currentTileID].Bank.PlayerBalance == 0)

{

PlayerManager.Instance.BanksWithFunds.RemoveAll(bankWF => bankWF.TileID == currentTileID);

bank.TimeTillAccountInterest = new MapTime(0, 0, 7, 0, 0);

}

UpdateBank();

WithdrawMoneyInput.ClearInputs();

}

}

public void TakeLoan()

{

if(duration != 0)

{

int loan = LoanMoneyInput.GetInputMoney();

Bank bank = MainManager.Instance.TileInfoMap[currentTileID].Bank;

MapTime durationLeft = new MapTime(0, duration, 0, 0, 0);

if (bank.VaultBalance >= loan)

{

int interestId = Mathf.RoundToInt(duration / 3);

PlayerManager.Instance.Loans.Add(new Loan(currentTileID, durationLeft, loan, bank.LoanInterest[interestId]));

MainManager.Instance.TileInfoMap[currentTileID].Bank.VaultBalance -= loan;

PlayerManager.Instance.Money += loan;

LoanMoneyInput.ClearInputs();

SelectDuration(0);

UpdateBank();

}

}

}

public void PayLoan()

{

Loan loan = PlayerManager.Instance.Loans.FirstOrDefault(loan => loan.BankTileID == currentTileID);

if (PlayerManager.Instance.Money >= loan.DebtLeft)

{

Bank bank = MainManager.Instance.TileInfoMap[currentTileID].Bank;

MainManager.Instance.TileInfoMap[currentTileID].Bank.VaultBalance += loan.DebtLeft;

PlayerManager.Instance.Money -= loan.DebtLeft;

PlayerManager.Instance.Loans.RemoveAll(bankLoan => bankLoan.BankTileID == currentTileID);

SelectDuration(0);

UpdateBank();

}

}

/\*

0 - Trade Buy

1 - Trade Sell

2 - Buildings

\*/

public void UpdateList(int filter)

{

UpdateStats();

GameObject listPrefab;

Dictionary<int, int> listInventory;

TileInfo tile = MainManager.Instance.TileInfoMap[currentTileID];

RectTransform contentPanel;

switch (filter)

{

case 0:

listPrefab = buyListRowPrefab;

listInventory = tile.TileInventory;

contentPanel = tradeContentPanel;

break;

case 1:

listPrefab = sellListRowPrefab;

listInventory = PlayerManager.Instance.Inventory;

contentPanel = tradeContentPanel;

break;

case 2:

listPrefab = buildingListRowPrefab;

listInventory = null;

contentPanel = buildingsContentPanel;

break;

default:

listPrefab = buyListRowPrefab;

listInventory = tile.TileInventory;

contentPanel = tradeContentPanel;

break;

}

foreach (Transform child in contentPanel)

{

Destroy(child.gameObject);

}

switch (filter)

{

case 0:

case 1:

foreach (KeyValuePair<int, int> entry in listInventory)

{

if (entry.Value > 0)

{

Item item = MainManager.Instance.Items[entry.Key];

GameObject newItem = Instantiate(listPrefab, contentPanel);

TextMeshProUGUI[] textComponents = newItem.GetComponentsInChildren<TextMeshProUGUI>();

int finalPrice = tile.TilePrices[item.Name];

if (filter == 0)

{

finalPrice = Mathf.RoundToInt(finalPrice \* buyModifier);

if (PlayerManager.Instance.Skills[0, 2])

{

finalPrice = Mathf.RoundToInt(finalPrice \* 0.95f);

}

finalPrice = Mathf.RoundToInt(finalPrice \* (1f - (PlayerManager.Instance.Attributes[0] / 1000f)));

if (finalPrice < 0)

{

finalPrice = 0;

}

}

else

{

if (PlayerManager.Instance.Skills[0, 1])

{

finalPrice = Mathf.RoundToInt(finalPrice \* 1.05f);

}

finalPrice = Mathf.RoundToInt(finalPrice \* (1f + (PlayerManager.Instance.Attributes[0] / 2000f)));

}

textComponents[0].text = item.Name;

textComponents[1].text = item.Description;

textComponents[2].text = entry.Value.ToString();

string localPrice = MainManager.Instance.MoneyToString(finalPrice);

if (finalPrice == item.BaseValue || !PlayerManager.Instance.Skills[0, 3])

{

textComponents[3].text = localPrice;

}

else if (PlayerManager.Instance.Skills[0, 3])

{

if (finalPrice > item.BaseValue)

{

textComponents[3].text = "<color=#089016>" + localPrice + "</color>";

}

else

{

textComponents[3].text = "<color=#C10303>" + localPrice + "</color>";

}

}

if (filter == 1)

{

Button sellBtn = newItem.transform.Find("Col 5").transform.Find("SellBtn").GetComponent<Button>();

sellBtn.onClick.AddListener(() => SellAction(entry.Key));

}

else if (filter == 0)

{

Button buyBtn = newItem.transform.Find("Col 5").transform.Find("BuyBtn").GetComponent<Button>();

buyBtn.onClick.AddListener(() => BuyAction(entry.Key));

}

}

}

break;

case 2:

{

foreach (Building building in tile.Buildings)

{

GameObject newItem = Instantiate(listPrefab, contentPanel);

TextMeshProUGUI[] textComponents = newItem.GetComponentsInChildren<TextMeshProUGUI>();

int buildingID = building.BuildingType;

BuildingType buildingType = MainManager.Instance.BuildingTypes[buildingID];

int finalPrice = Mathf.RoundToInt(buildingType.BaseValue \* building.ValueMod);

if (building.PlayerOwned)

{

finalPrice = Mathf.RoundToInt(finalPrice \* (1f + (PlayerManager.Instance.Attributes[0] / 2000f)));

}

else

{

finalPrice = Mathf.RoundToInt(finalPrice \* buyModifier);

finalPrice = Mathf.RoundToInt(finalPrice \* (1f - (PlayerManager.Instance.Attributes[0] / 1000f)));

if (finalPrice < 0)

{

finalPrice = 0;

}

}

float finalIncomeMod = building.IncomeMod;

if(PlayerManager.Instance.Skills[2, 4])

{

finalIncomeMod += 0.25f;

}

int finalIncome = Mathf.RoundToInt(buildingType.Income \* finalIncomeMod);

string color = "FFEECC";

if (building.PlayerOwned)

{

color = "FFC800";

}

textComponents[0].text = "<color=#" + color + ">" + buildingType.Name + "</color>";

textComponents[1].text = "<color=#" + color + ">" + buildingType.Description + "</color>";

textComponents[2].text = "<color=#" + color + ">" + MainManager.Instance.MoneyToString(finalIncome) + "</color>";

textComponents[3].text = "<color=#" + color + ">" + MainManager.Instance.MoneyToString(finalPrice) + "</color>";

Button Btn = newItem.transform.Find("Col 5/ActionBtn").GetComponent<Button>();

TextMeshProUGUI btnText = newItem.transform.Find("Col 5/ActionBtn").GetComponentInChildren<TextMeshProUGUI>();

if (building.PlayerOwned)

{

btnText.text = "<color=#" + color + ">Sell</color>";

Btn.onClick.AddListener(() => SellBuilding(building));

}

else

{

btnText.text = "<color=#" + color + ">Buy</color>";

Btn.onClick.AddListener(() => BuyBuilding(building));

}

}

}

break;

default:

break;

}

}

void BuyBuilding(Building building)

{

int id = building.BuildingInTile;

int tileID = currentTileID;

Building[] buildings = MainManager.Instance.TileInfoMap[tileID].Buildings;

long baseValue = MainManager.Instance.BuildingTypes[building.BuildingType].BaseValue;

long price = Mathf.RoundToInt(baseValue \* building.ValueMod);

price = Mathf.RoundToInt(price \* buyModifier);

price = Mathf.RoundToInt(price \* (1f - (PlayerManager.Instance.Attributes[0] / 1000f)));

if (price < 0)

{

price = 0;

}

if (PlayerManager.Instance.Money >= price)

{

PlayerManager.Instance.Money -= price;

buildings[id].PlayerOwned = true;

PlayerManager.Instance.Buildings.Add(building);

if (price < baseValue)

{

PlayerManager.Instance.XP += baseValue - price;

PlayerManager.Instance.UpdateLevel();

}

MainManager.Instance.TileInfoMap[tileID].TileWealth[5] += price; // Todo: Replace with function that updates all wealth

MainManager.Instance.TileInfoMap[tileID].Buildings = buildings;

UpdateList(2);

}

}

void SellBuilding(Building building)

{

int id = building.BuildingInTile;

int tileID = currentTileID;

Building[] buildings = MainManager.Instance.TileInfoMap[tileID].Buildings;

long baseValue = MainManager.Instance.BuildingTypes[building.BuildingType].BaseValue;

long price = Mathf.RoundToInt(baseValue \* building.ValueMod);

price = Mathf.RoundToInt(price \* (1f + (PlayerManager.Instance.Attributes[0] / 2000f)));

PlayerManager.Instance.Money += price;

buildings[id].PlayerOwned = false;

PlayerManager.Instance.Buildings.RemoveAll(b => b.TileID == tileID && b.BuildingInTile == id);

if (price > baseValue)

{

PlayerManager.Instance.XP += price - baseValue;

PlayerManager.Instance.UpdateLevel();

}

MainManager.Instance.TileInfoMap[tileID].TileWealth[5] -= price; // Todo: Replace with function that updates all wealth

MainManager.Instance.TileInfoMap[tileID].Buildings = buildings;

UpdateList(2);

}

void BuyAction(int itemID)

{

string itemName = MainManager.Instance.Items[itemID].Name;

int tileID = currentTileID;

TileInfo tile = MainManager.Instance.TileInfoMap[tileID];

long price = Mathf.RoundToInt(tile.TilePrices[itemName] \* buyModifier);

if(PlayerManager.Instance.Skills[0, 2])

{

price = Mathf.RoundToInt(price \* 0.95f);

}

price = Mathf.RoundToInt(price \* (1f - (PlayerManager.Instance.Attributes[0] / 1000f)));

if (price < 0)

{

price = 0;

}

if (tile.TileInventory.ContainsKey(itemID) && tile.TileInventory[itemID] > 0)

{

if (PlayerManager.Instance.Money >= price)

{

PlayerManager.Instance.Money -= price;

if (!PlayerManager.Instance.Inventory.ContainsKey(itemID))

{

PlayerManager.Instance.Inventory[itemID] = 0;

}

PlayerManager.Instance.Inventory[itemID] += 1;

tile.TileInventory[itemID] -= 1;

if(price < MainManager.Instance.Items[itemID].BaseValue)

{

PlayerManager.Instance.XP += MainManager.Instance.Items[itemID].BaseValue - price;

PlayerManager.Instance.UpdateLevel();

}

tile.TileWealth[5] += price; // Todo: Replace with function that updates all wealth

MainManager.Instance.TileInfoMap[tileID] = tile;

UpdateList(0);

}

}

}

void SellAction(int itemID)

{

string itemName = MainManager.Instance.Items[itemID].Name;

int tileID = currentTileID;

TileInfo tile = MainManager.Instance.TileInfoMap[tileID];

long price = tile.TilePrices[itemName];

if (PlayerManager.Instance.Skills[0, 1])

{

price = Mathf.RoundToInt(price \* 1.05f);

}

price = Mathf.RoundToInt(price \* (1f + (PlayerManager.Instance.Attributes[0] / 2000f)));

if (PlayerManager.Instance.Inventory.ContainsKey(itemID))

{

if (PlayerManager.Instance.Inventory[itemID] > 0)

{

PlayerManager.Instance.Money += price;

if (price > MainManager.Instance.Items[itemID].BaseValue)

{

PlayerManager.Instance.XP += price - MainManager.Instance.Items[itemID].BaseValue;

PlayerManager.Instance.UpdateLevel();

}

PlayerManager.Instance.Inventory[itemID] -= 1;

tile.TileInventory[itemID] += 1;

tile.TileWealth[5] -= price; // Todo: Replace with function that updates all wealth

MainManager.Instance.TileInfoMap[tileID] = tile;

UpdateList(1);

}

}

}

public void HideAllTabs()

{

StatsTab.SetActive(false);

TradeTab.SetActive(false);

BuildingsTab.SetActive(false);

BankTab.SetActive(false);

CaravansTab.SetActive(false);

BuySellBtns.SetActive(false);

}

}

**InventoryMenu.cs**

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

using UnityEngine.UI;

using TMPro;

public class InventoryMenu : MonoBehaviour

{

public GameObject listRowPrefab;

public RectTransform contentPanel;

void Start()

{

UpdateList(0);

LayoutRebuilder.ForceRebuildLayoutImmediate(contentPanel);

float contentHeight = contentPanel.rect.height;

contentPanel.anchoredPosition = new Vector2(contentPanel.anchoredPosition.x, -(contentHeight / 2));

}

public void UpdateList(int filter)

{

foreach (Transform child in contentPanel)

{

Destroy(child.gameObject);

}

foreach (KeyValuePair<int, int> entry in PlayerManager.Instance.Inventory)

{

if(entry.Value > 0)

{

Item item = MainManager.Instance.Items[entry.Key];

if (filter == 0 || filter == item.Type)

{

GameObject newItem = Instantiate(listRowPrefab, contentPanel);

TextMeshProUGUI[] textComponents = newItem.GetComponentsInChildren<TextMeshProUGUI>();

textComponents[0].text = item.Name;

textComponents[1].text = item.Description;

textComponents[2].text = entry.Value.ToString();

textComponents[3].text = MainManager.Instance.MoneyToString(item.BaseValue);

string localPrice = MainManager.Instance.MoneyToString(item.LocalPrice);

if (item.LocalPrice > item.BaseValue)

{

textComponents[4].text = "<color=#089016>" + localPrice + "</color>";

}

else if (item.LocalPrice < item.BaseValue)

{

textComponents[4].text = "<color=#C10303>" + localPrice + "</color>";

}

else

{

textComponents[4].text = localPrice;

}

Button dropBtn = newItem.transform.Find("Col 6").transform.Find("DropBtn").GetComponent<Button>();

dropBtn.onClick.AddListener(() => DropAction(entry.Key, filter));

}

}

}

}

public void DropAction(int ItemID, int filter)

{

if (PlayerManager.Instance.Inventory[ItemID] > 0)

{

PlayerManager.Instance.Inventory[ItemID] -= 1;

}

UpdateList(filter);

}

void Update()

{

}

}

**MainManager.cs**

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

using System.IO;

using System.Linq;

using UnityEngine.Tilemaps;

using System.Runtime.Serialization.Formatters.Binary;

using UnityEngine.UI;

[System.Serializable]

public class TileInfo

{

public string TileName { get; set; }

public int[] TilePop { get; set; }

public long[] TileWealth { get; set; }

public int[] TileFood { get; set; }

public int TileMoveCost { get; set; }

public Dictionary<int, int> TileInventory { get; set; }

public Dictionary<string, int> TilePrices { get; set; }

public Building[] Buildings { get; set; }

public Bank Bank { get; set; }

public TileInfo(string name, int[] pop, long[] wealth, int[] food, int tileMoveCost, Dictionary<int, int> tileInventory, Dictionary<string, int> tilePrices, Building[] buildings, Bank bank)

{

TileName = name;

TilePop = pop;

TileWealth = wealth;

TileFood = food;

TileMoveCost = tileMoveCost;

TileInventory = tileInventory;

TilePrices = tilePrices;

Buildings = buildings;

Bank = bank;

}

}

public class TileGenerator

{

public string[] TileNames { get; set; }

public int TilePopMin { get; set; }

public int TilePopMax { get; set; }

public int TileWealthMin { get; set; }

public int TileWealthMax { get; set; }

public int TileFoodMin { get; set; }

public int TileFoodMax { get; set; }

public int TileMoveCost { get; set; }

public Dictionary<string, int> TileInventoryMin { get; set; }

public Dictionary<string, int> TileInventoryMax { get; set; }

public Dictionary<string, int> TilePricesMin { get; set; }

public Dictionary<string, int> TilePricesMax { get; set; }

public TileGenerator(

string[] tileNames,

int tilePopMin, int tilePopMax,

int tileWealthMin, int tileWealthMax,

int tileFoodMin, int tileFoodMax,

int tileMoveCost,

Dictionary<string, int> tileInventoryMin, Dictionary<string, int> tileInventoryMax,

Dictionary<string, int> tilePricesMin, Dictionary<string, int> tilePricesMax

)

{

TileNames = tileNames;

TilePopMin = tilePopMin;

TilePopMax = tilePopMax;

TileWealthMin = tileWealthMin;

TileWealthMax = tileWealthMax;

TileFoodMin = tileFoodMin;

TileFoodMax = tileFoodMax;

TileMoveCost = tileMoveCost;

TileInventoryMin = tileInventoryMin;

TileInventoryMax = tileInventoryMax;

TilePricesMin = tilePricesMin;

TilePricesMax = tilePricesMax;

}

}

[System.Serializable]

public class SerializableVector3Int

{

public int x;

public int y;

public int z;

public SerializableVector3Int(Vector3Int vec)

{

x = vec.x;

y = vec.y;

z = vec.z;

}

public Vector3Int ToVector3Int()

{

return new Vector3Int(x, y, z);

}

}

[System.Serializable]

public class SerializableVector3

{

public float x;

public float y;

public float z;

public SerializableVector3(Vector3 vec)

{

x = vec.x;

y = vec.y;

z = vec.z;

}

public Vector3 ToVector3()

{

return new Vector3(x, y, z);

}

}

[System.Serializable]

public class MapData

{

public MapTime GlobalTime;

public Dictionary<SerializableVector3Int, int> TileIDMap;

public Dictionary<int, TileInfo> TileInfoMap;

}

[System.Serializable]

public class Item

{

public long BaseValue;

public string Name;

public string Description;

public int Type;

public long LocalPrice;

public Item(int baseValue, string name, string description, int type)

{

BaseValue = baseValue;

Name = name;

Description = description;

Type = type;

LocalPrice = BaseValue;

}

/\*

Types:

- 0 = All

- 1 = Equipment

- 2 = Food

- 3 = Resources

- 4 = Transport

\*/

}

[System.Serializable]

public class BuildingType

{

public long BaseValue;

public string Name;

public string Description;

public int Type;

public long LocalPrice;

public long Income;

public BuildingType(int baseValue, string name, string description, int income)

{

BaseValue = baseValue;

Name = name;

Description = description;

Income = income;

LocalPrice = BaseValue;

}

}

[System.Serializable]

public class Building

{

public int BuildingInTile;

public int BuildingType;

public int TileID;

public bool PlayerOwned;

public float ValueMod;

public float IncomeMod;

public Building(int buildingInTile, int buildingType, int tileID)

{

BuildingInTile = buildingInTile;

BuildingType = buildingType;

TileID = tileID;

PlayerOwned = false;

ValueMod = Random.Range(0.75f, 1.5f);

IncomeMod = Random.Range(0.75f, 1.5f);

}

}

[System.Serializable]

public class Bank

{

public int TileID;

public long VaultBalance;

public int[] LoanInterest;

public int AccountInterest;

public long PlayerBalance;

public MapTime TimeTillAccountInterest;

public Bank(int tileID)

{

TileID = tileID;

VaultBalance = Random.Range(10, 1500000000);

LoanInterest = new int[3];

LoanInterest[0] = Random.Range(10, 15);

LoanInterest[1] = Random.Range(LoanInterest[0], LoanInterest[0] + 10);

LoanInterest[2] = Random.Range(LoanInterest[1], LoanInterest[1] + 10);

AccountInterest = Random.Range(5, 20);

PlayerBalance = 0;

TimeTillAccountInterest = new MapTime(0, 0, 7, 0, 0);

}

}

[System.Serializable]

public class Loan

{

public int BankTileID;

public MapTime DurationLeft;

public long DebtLeft;

public float Interest;

public Loan(int tileID, MapTime durationLeft, int debtLeft, float interest)

{

BankTileID = tileID;

DurationLeft = durationLeft;

DebtLeft = debtLeft;

Interest = interest;

}

public float InterestToFloat()

{

return Interest \* 0.01f;

}

}

[System.Serializable]

public class MapTime

{

public int Years;

public int Months;

public int Days;

public int Hours;

public int Minutes;

public MapTime( int years, int months, int days, int hours, int minutes)

{

Years = years;

Months = months;

Days = days;

Hours = hours;

Minutes = minutes;

UpdateTime();

}

public int[] UpdateTime(int years = 0, int months = 0, int days = 0, int hours = 0, int minutes = 0)

{

int OldYears = Years;

int OldMonths = Months;

int OldDays = Days;

int OldHours = Hours;

int OldMinutes = Minutes;

Years += years;

Months += months;

Days += days;

Hours += hours;

Minutes += minutes;

while (Minutes > 59 || Hours > 23 || Days > 29 || Months > 11)

{

if (Hours > 23)

{

Hours -= 24;

Days += 1;

}

if (Days > 29)

{

Days -= 30;

Months += 1;

}

if (Months > 11)

{

Months -= 12;

Years += 1;

}

}

while (Years < 0 || Months < 0 || Days < 0 || Hours < 0 || Minutes < 0)

{

if (Years < 0)

{

Years = 0;

Months = 0;

Days = 0;

Hours = 0;

}

else

{

if (Minutes < 0)

{

Hours -= 1;

Minutes += 60;

}

if (Hours < 0)

{

Days -= 1;

Hours += 24;

}

if (Days < 0)

{

Days += 30;

Months -= 1;

}

if (Months < 0)

{

Months += 12;

Years -= 1;

}

}

}

return new int[]{ Years - OldYears, Months - OldMonths, Days - OldDays, Hours - OldHours, Minutes - OldMinutes };

}

public bool ZeroTime()

{

return Years == 0 && Months == 0 && Days == 0 && Hours == 0;

}

public int ToDays()

{

int days = 0;

days += Years \* 12 \* 30;

days += Months \* 30;

days += Days;

return days;

}

public string TimeToString()

{

string result = "";

if( Years > 0 )

{

result += Years.ToString() + "Y ";

}

if( Months > 0 )

{

result += Months.ToString() + "M ";

}

if( Days > 0 )

{

result += Days.ToString() + "D ";

}

if( Hours > 0 )

{

result += Hours.ToString() + "H ";

}

result.Remove(result.Length - 1);

return result;

}

}

[System.Serializable]

public class Quest

{

public string Name;

public string Description;

public int Type;

public MapTime Expires;

public int Reward;

public bool Tracked;

public bool Completed;

public List<int> Progress;

public int TriggerID;

public Quest(string name, string description, int type, MapTime expires, int reward, int triggerID)

{

Name = name;

Description = description;

Type = type;

Expires = expires;

Reward = reward;

Tracked = false;

Completed = false;

Progress = new List<int> { 0 };

TriggerID = triggerID;

}

public string TypeString()

{

switch (Type)

{

case 0:

return "Main";

case 1:

return "Side";

case 2:

return "Town";

default:

return "Other";

}

}

public void ProgressUpdate(int index, int value)

{

if (index < Progress.Count)

{

Progress[index] += value;

}

else

{

while (Progress.Count < index)

{

Progress.Add(0);

}

Progress.Add(value);

}

}

public void ProgressSet(int index, int value)

{

if (index < Progress.Count)

{

Progress[index] = value;

}

else

{

while (Progress.Count < index)

{

Progress.Add(0);

}

Progress.Add(value);

}

}

}

public class MainManager : MonoBehaviour

{

public static MainManager Instance;

public AudioSource audioSource;

public string VolumePrefsKey = "VolumeLevel"; // Key for PlayerPrefs

public Dictionary<string, TileGenerator> TileGeneratorMap = new Dictionary<string, TileGenerator>();

public Item[] Items = new Item[]{

new Item(15, "Wood", "Just some wood", 3),

new Item(25, "Iron", "Regular iron ingots", 3),

new Item(30, "Coal", "Ordinry lumps of coal", 3),

new Item(10, "Leather", "Untanned leather", 3),

new Item(75, "Crystals", "Very shiny crystals", 3),

new Item(5, "Cloth", "Cloth of unknow origin", 3),

new Item(300, "Weapons", "Various weapons of war", 1),

new Item(1250, "Potions", "Mystical potions", 1),

new Item(3, "Grain", "Most basic food", 2)

};

public BuildingType[] BuildingTypes = new BuildingType[]

{

new BuildingType(500, "Bakery", "Bakes bread and pastries.", 50),

new BuildingType(1000, "Blacksmith", "Produces weapons and tools.", 100),

new BuildingType(200, "Butcher Shop", "Processes and sells meat.", 30),

new BuildingType(3000, "Tavern", "Serves food and drink.", 300),

new BuildingType(5000, "Inn", "Provides lodging for travelers.", 500),

new BuildingType(2500, "Market", "Central trading hub for goods.", 250),

new BuildingType(700, "Farm", "Produces various crops.", 70),

new BuildingType(1200, "Mill", "Processes grains into flour.", 120),

new BuildingType(1500, "Stable", "Houses and trades horses.", 150),

new BuildingType(2200, "Tailor Shop", "Produces and sells clothing.", 220)

};

public bool GameLoaded = false;

// Saveable

public MapTime GlobalTime = new MapTime(1333, 0, 0, 12, 0);

public Dictionary<Vector3Int, int> TileIDMap = new Dictionary<Vector3Int, int>();

public Dictionary<int, TileInfo> TileInfoMap = new Dictionary<int, TileInfo>();

public void SaveMapData(string saveName)

{

BinaryFormatter bf = new BinaryFormatter();

FileStream file = File.Create(Application.persistentDataPath + "/saves/map" + saveName + ".sav");

// Convert TileIDMap keys to SerializableVector3Int

Dictionary<SerializableVector3Int, int> serializableTileIDMap = new Dictionary<SerializableVector3Int, int>();

foreach (var kvp in TileIDMap)

{

serializableTileIDMap[new SerializableVector3Int(kvp.Key)] = kvp.Value;

}

MapData mapData = new MapData

{

GlobalTime = GlobalTime,

TileIDMap = serializableTileIDMap,

TileInfoMap = TileInfoMap

};

bf.Serialize(file, mapData);

file.Close();

Debug.Log("Saved Map");

}

// Function to load map data from a file

public bool LoadMapData(string saveName)

{

string path = Application.persistentDataPath + "/saves/map" + saveName + ".sav";

if (File.Exists(path))

{

BinaryFormatter bf = new BinaryFormatter();

FileStream file = File.Open(Application.persistentDataPath + "/saves/map" + saveName + ".sav", FileMode.Open);

MapData mapData = (MapData)bf.Deserialize(file);

file.Close();

// Convert TileIDMap keys back to Vector3Int

Dictionary<Vector3Int, int> deserializedTileIDMap = new Dictionary<Vector3Int, int>();

foreach (var kvp in mapData.TileIDMap)

{

deserializedTileIDMap[kvp.Key.ToVector3Int()] = kvp.Value;

}

GlobalTime = mapData.GlobalTime;

TileIDMap = deserializedTileIDMap;

TileInfoMap = mapData.TileInfoMap;

Debug.Log("Loaded Map");

return true;

}

else

{

Debug.Log("Not Loaded Map");

return false;

}

}

public void NewGame()

{

Items = new Item[]{

new Item(15, "Wood", "Just some wood", 3),

new Item(25, "Iron", "Regular iron ingots", 3),

new Item(30, "Coal", "Ordinary lumps of coal", 3),

new Item(10, "Leather", "Untanned leather", 3),

new Item(75, "Crystals", "Very shiny crystals", 3),

new Item(5, "Cloth", "Cloth of unknow origin", 3),

new Item(300, "Weapons", "Various weapons of war", 1),

new Item(1250, "Potions", "Mystical potions", 1),

new Item(3, "Grain", "Most basic food", 2)

};

PlayerManager.Instance.Quests.Add(new Quest("New People", "Enter any town", 0, new MapTime(1,0,0,0, 0), 10, 0));

}

public void SaveGame()

{

PlayerManager.Instance.SavePlayerData("Test0");

SaveMapData("Test0");

}

public void LoadGame()

{

if (PlayerManager.Instance.LoadPlayerData("Test0") && LoadMapData("Test0"))

{

GameLoaded = true;

}

else {

NewGame();

}

}

private void Init()

{

GenerateTileGeneratorMap();

if (PlayerPrefs.HasKey(VolumePrefsKey))

{

audioSource.volume = PlayerPrefs.GetFloat(VolumePrefsKey);

}

}

private void GenerateTileGeneratorMap()

{

TileGeneratorMap["Tile\_Grass\_Empty\_Plains"] = new TileGenerator(

new string[] { "Grass" },

0,

0,

0,

0,

0,

0,

10,

new Dictionary<string, int> {},

new Dictionary<string, int> {},

new Dictionary<string, int> {},

new Dictionary<string, int> {}

);

TileGeneratorMap["Grass\_DirtRoad\_NE"] = new TileGenerator(

new string[] { "Road" },

0,

0,

0,

0,

0,

0,

5,

new Dictionary<string, int> { },

new Dictionary<string, int> { },

new Dictionary<string, int> { },

new Dictionary<string, int> { }

);

TileGeneratorMap["Grass\_DirtRoad\_NS"] = new TileGenerator(

new string[] { "Road" },

0,

0,

0,

0,

0,

0,

5,

new Dictionary<string, int> { },

new Dictionary<string, int> { },

new Dictionary<string, int> { },

new Dictionary<string, int> { }

);

TileGeneratorMap["Grass\_DirtRoad\_NW"] = new TileGenerator(

new string[] { "Road" },

0,

0,

0,

0,

0,

0,

5,

new Dictionary<string, int> { },

new Dictionary<string, int> { },

new Dictionary<string, int> { },

new Dictionary<string, int> { }

);

TileGeneratorMap["Grass\_DirtRoad\_ES"] = new TileGenerator(

new string[] { "Road" },

0,

0,

0,

0,

0,

0,

5,

new Dictionary<string, int> { },

new Dictionary<string, int> { },

new Dictionary<string, int> { },

new Dictionary<string, int> { }

);

TileGeneratorMap["Grass\_DirtRoad\_EW"] = new TileGenerator(

new string[] { "Road" },

0,

0,

0,

0,

0,

0,

5,

new Dictionary<string, int> { },

new Dictionary<string, int> { },

new Dictionary<string, int> { },

new Dictionary<string, int> { }

);

TileGeneratorMap["Grass\_DirtRoad\_SW"] = new TileGenerator(

new string[] { "Road" },

0,

0,

0,

0,

0,

0,

5,

new Dictionary<string, int> { },

new Dictionary<string, int> { },

new Dictionary<string, int> { },

new Dictionary<string, int> { }

);

TileGeneratorMap["Grass\_DirtRoad\_NES"] = new TileGenerator(

new string[] { "Road" },

0,

0,

0,

0,

0,

0,

5,

new Dictionary<string, int> { },

new Dictionary<string, int> { },

new Dictionary<string, int> { },

new Dictionary<string, int> { }

);

TileGeneratorMap["Grass\_DirtRoad\_NSW"] = new TileGenerator(

new string[] { "Road" },

0,

0,

0,

0,

0,

0,

5,

new Dictionary<string, int> { },

new Dictionary<string, int> { },

new Dictionary<string, int> { },

new Dictionary<string, int> { }

);

TileGeneratorMap["Grass\_DirtRoad\_NEW"] = new TileGenerator(

new string[] { "Road" },

0,

0,

0,

0,

0,

0,

5,

new Dictionary<string, int> { },

new Dictionary<string, int> { },

new Dictionary<string, int> { },

new Dictionary<string, int> { }

);

TileGeneratorMap["Grass\_DirtRoad\_ESW"] = new TileGenerator(

new string[] { "Road" },

0,

0,

0,

0,

0,

0,

5,

new Dictionary<string, int> { },

new Dictionary<string, int> { },

new Dictionary<string, int> { },

new Dictionary<string, int> { }

);

TileGeneratorMap["Grass\_DirtRoad\_NESW"] = new TileGenerator(

new string[] { "Road" },

0,

0,

0,

0,

0,

0,

5,

new Dictionary<string, int> { },

new Dictionary<string, int> { },

new Dictionary<string, int> { },

new Dictionary<string, int> { }

);

TileGeneratorMap["Grass\_Forrest"] = new TileGenerator(

new string[] { "Forrest" },

0,

0,

0,

0,

0,

0,

15,

new Dictionary<string, int> { },

new Dictionary<string, int> { },

new Dictionary<string, int> { },

new Dictionary<string, int> { }

);

TileGeneratorMap["Grass\_Hill"] = new TileGenerator(

new string[] { "Hill" },

0,

0,

0,

0,

0,

0,

20,

new Dictionary<string, int> { },

new Dictionary<string, int> { },

new Dictionary<string, int> { },

new Dictionary<string, int> { }

);

TileGeneratorMap["Tile\_Grass\_City\_2-2-Dirt"] = new TileGenerator(

new string[] { "Aldermoor","Briarwick","Crestmoor","Dunmere","Elderford","Fairholm","Goldfield","Hawthorne","Ivyport","Juniper's Crossing","Kestrel's Reach","Larkspur","Meadowbrook","Nightingale","Oakhaven","Pinecrest","Quail's Rest","Rosewater","Sagefield","Thistlebrook","Umberwood","Valleyview","Willowgate","Yarrowfell","Zephyr's Hollow","Auburnstead","Bramblebury","Copperwood","Dewhurst","Embermoor","Fernwood","Glenhaven","Hazelridge","Irisvale","Jade Hollow","Kingsford","Lilyvale","Mossridge","Noble's Landing","Oakenshade","Primrose Hill","Quartzvale","Raven's Nest","Silverfall","Tanglewood","Umberton","Verdant Hollow","Wispwater","Xanadu's Rest"},

100,

9999,

1000,

99999,

10,

100,

5,

new Dictionary<string, int> { { "Wood", 5 },{ "Iron", 5 },{ "Coal", 5 },{ "Leather", 5 },{ "Crystals", 0 },{ "Cloth", 5 },{ "Weapons", 0 },{ "Potions", 0 }, { "Grain", 10 } },

new Dictionary<string, int> { { "Wood", 100 },{ "Iron", 100 },{ "Coal", 100 },{ "Leather", 100 },{ "Crystals", 25 },{ "Cloth", 100 },{ "Weapons", 50 },{ "Potions", 50 }, { "Grain", 100 } },

new Dictionary<string, int> { { "Wood", 10 },{ "Iron", 15 },{ "Coal", 12 },{ "Leather", 7 },{ "Crystals", 40 },{ "Cloth", 3 },{ "Weapons", 20 },{ "Potions", 100 }, { "Grain", 1 } },

new Dictionary<string, int> { { "Wood", 20 },{ "Iron", 30 },{ "Coal", 40 },{ "Leather", 25 },{ "Crystals", 100 },{ "Cloth", 10 },{ "Weapons", 500 },{ "Potions", 2400 }, { "Grain", 5 } }

);

}

public void GenerateTileIDs(Tilemap tilemap)

{

int currentID = 1;

foreach (var pos in tilemap.cellBounds.allPositionsWithin)

{

if (tilemap.HasTile(pos))

{

TileIDMap[pos] = currentID;

TileBase currentTile = tilemap.GetTile(pos);

string tileName = currentTile.name;

TileGenerator currentTileGenerator = TileGeneratorMap[tileName];

string randomName = GenerateRandomName(tileName);

int tileMoveCost = currentTileGenerator.TileMoveCost;

int[] randomPop = new int[6], randomFood = new int[6];

long[] randomWealth = new long[6];

Dictionary<int, int> randomInventory = null;

Dictionary<string, int> randomPriceList = null;

Building[] randomBuildings = null;

if (currentTile != null && currentTile.name.Contains("City"))

{

randomPop = GenerateInfoArray(currentTileGenerator.TilePopMin, currentTileGenerator.TilePopMax);

int[] rW = GenerateInfoArray(currentTileGenerator.TileWealthMin, currentTileGenerator.TileWealthMax);

for (int i = 0; i < rW.Length; i++)

{

randomWealth[i] = rW[i];

}

randomFood = GenerateInfoArray(currentTileGenerator.TileFoodMin, currentTileGenerator.TileFoodMax);

randomInventory = GenerateRandomInventory(tileName);

randomPriceList = GenerateRandomPriceList(tileName);

randomBuildings = GenerateRandomBuildings(currentID);

}

TileInfo tileInfo = new TileInfo(randomName, randomPop, randomWealth, randomFood, tileMoveCost, randomInventory ?? new Dictionary<int, int>(), randomPriceList ?? new Dictionary<string, int>(), randomBuildings ?? new Building[1], new Bank(currentID));

TileInfoMap[currentID] = tileInfo;

currentID++;

}

}

}

public Building[] GenerateRandomBuildings(int tileID)

{

int buildingCount = Random.Range(3, 13);

Building[] result = new Building[buildingCount];

for (int i = 0; i < result.Length; i++)

{

result[i] = new Building(i ,Random.Range(1, BuildingTypes.Length) - 1, tileID);

}

return result;

}

public int[] GenerateInfoArray(int min, int max)

{

int[] result = new int[6];

for (int i = 0; i < 6; i++)

{

result[i] = UnityEngine.Random.Range(min, max);

}

return result;

}

public string GenerateRandomName(string tileName)

{

string[] tileNames = TileGeneratorMap[tileName].TileNames;

return tileNames[UnityEngine.Random.Range(0, tileNames.Length)];

}

public Dictionary<int, int> GenerateRandomInventory(string tileName)

{

TileGenerator curTileGen = TileGeneratorMap[tileName];

Dictionary<int, int> inventory = new Dictionary<int, int>();

int i = 0;

foreach (Item item in Items)

{

inventory[i] = UnityEngine.Random.Range(curTileGen.TileInventoryMin[item.Name], curTileGen.TileInventoryMax[item.Name]);

i++;

}

return inventory;

}

public Dictionary<string, int> GenerateRandomPriceList(string tileName)

{

TileGenerator curTileGen = TileGeneratorMap[tileName];

Dictionary<string, int> priceList = new Dictionary<string, int>();

foreach (Item item in Items)

{

priceList[item.Name] = UnityEngine.Random.Range(curTileGen.TilePricesMin[item.Name], curTileGen.TilePricesMax[item.Name]);

}

return priceList;

}

public string MoneyToString(long money)

{

int moneyToSilver = 100;

int silverToGold = 100;

int goldToPlatinum = 100;

// Calculate each denomination

long absMoney = System.Math.Abs(money);

int bronze = (int)(absMoney % moneyToSilver);

int silver = (int)((absMoney / moneyToSilver) % silverToGold);

int gold = (int)((absMoney / (moneyToSilver \* silverToGold)) % goldToPlatinum);

long platinum = absMoney / (moneyToSilver \* silverToGold \* goldToPlatinum);

// Build the result string

string result = "";

if (platinum != 0)

result += platinum + "P ";

if (gold != 0)

result += gold + "G ";

if (silver != 0)

result += silver + "S ";

if (bronze != 0 || result == "")

result += bronze + "B";

// Add the negative sign if the original money value was negative

if (money < 0)

{

result = "-" + result.Trim();

}

return result.Trim();

}

public int GetTileID(Vector3Int position)

{

if (TileIDMap.ContainsKey(position))

{

return TileIDMap[position];

}

return -1;

}

public void PlotGraph(RectTransform graphPanel, float[] values, float dotScale, Sprite circleSprite, float margin, float lineWidth, Color positiveChangeColor, Color negativeChangeColor)

{

foreach (Transform child in graphPanel)

{

Destroy(child.gameObject);

}

margin = margin + dotScale;

float graphWidth = graphPanel.rect.width - 2 \* margin;

float graphHeight = graphPanel.rect.height - 2 \* margin;

float xSpacing = graphWidth / (values.Length - 1); // Space between points on the x-axis

float maxValue = Mathf.Max(values); // Find the maximum value to normalize y-values

float minValue = Mathf.Min(values); // Find the minimum value to normalize y-values

float yRange = maxValue - minValue;

if (yRange == 0)

{

yRange = 1;

}

float yMultiplier = graphHeight / yRange; // Scale y-values to fit the graph height

Vector3[] positions = new Vector3[values.Length];

for (int i = 0; i < values.Length; i++)

{

float x = -graphWidth / 2 + i \* xSpacing;

float normalizedY = (values[i] - minValue) / yRange; // Normalize y value to be between 0 and 1

float y = normalizedY \* graphHeight - graphHeight / 2; // Scale to graph height and center

positions[i] = new Vector3(x, y, 0);

if (i > 0)

{

// Determine color based on value change

Color color = values[i] >= values[i - 1] ? positiveChangeColor : negativeChangeColor;

CreateLine(positions[i - 1], positions[i], color, graphPanel, lineWidth);

}

}

// Create dots after creating lines

for (int i = 0; i < values.Length; i++)

{

if (i == 0)

{

Color color = values[i + 1] >= values[i] ? positiveChangeColor : negativeChangeColor;

CreateDot(positions[i], color, graphPanel, dotScale, circleSprite);

}

else

{

Color color = values[i] >= values[i - 1] ? positiveChangeColor : negativeChangeColor;

CreateDot(positions[i], color, graphPanel, dotScale, circleSprite);

}

}

}

void CreateDot(Vector3 position, Color color, RectTransform graphPanel, float dotScale, Sprite circleSprite)

{

GameObject dot = new GameObject("Dot", typeof(Image));

dot.transform.SetParent(graphPanel, false);

Image dotImage = dot.GetComponent<Image>();

dotImage.sprite = circleSprite;

dotImage.color = color;

RectTransform rectTransform = dot.GetComponent<RectTransform>();

rectTransform.anchoredPosition = new Vector2(position.x, position.y);

rectTransform.sizeDelta = new Vector2(dotScale, dotScale);

}

void CreateLine(Vector3 start, Vector3 end, Color color, RectTransform graphPanel, float lineWidth)

{

GameObject lineObject = new GameObject("Line", typeof(Image));

lineObject.transform.SetParent(graphPanel, false);

Image lineImage = lineObject.GetComponent<Image>();

lineImage.color = color;

RectTransform rectTransform = lineObject.GetComponent<RectTransform>();

rectTransform.anchorMin = rectTransform.anchorMax = new Vector2(0.5f, 0.5f);

rectTransform.sizeDelta = new Vector2(Vector3.Distance(start, end), lineWidth);

rectTransform.pivot = new Vector2(0, 0.5f);

rectTransform.anchoredPosition = new Vector2(start.x, start.y);

rectTransform.rotation = Quaternion.FromToRotation(Vector3.right, end - start);

}

private void Awake()

{

if (Instance != null)

{

Destroy(gameObject);

return;

}

Init();

Instance = this;

DontDestroyOnLoad(gameObject);

}

}

**MainMenu.cs**

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

using UnityEngine.SceneManagement;

public class MainMenu : MonoBehaviour

{

public GameObject ContinueButton;

private void Update()

{

ContinueButton.SetActive(MainManager.Instance.GameLoaded);

}

public void PlayGame()

{

MainManager.Instance.NewGame();

SceneManager.LoadSceneAsync("CharacterCreation");

}

public void ContinueGame()

{

SceneManager.LoadSceneAsync("MapView");

PlayerManager.Instance.LoadPlayerPosition();

}

public void LoadGame()

{

MainManager.Instance.LoadGame();

SceneManager.LoadSceneAsync("MapView");

PlayerManager.Instance.LoadPlayerPosition();

}

public void ExitGame()

{

Application.Quit();

}

}

**MapMenu.cs**

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

using UnityEngine.SceneManagement;

public class MapMenu : MonoBehaviour

{

public void MainMenu()

{

SceneManager.LoadSceneAsync("MainMenu");

}

public void SaveGame()

{

MainManager.Instance.SaveGame();

}

public void LoadGame()

{

MainManager.Instance.LoadGame();

PlayerManager.Instance.LoadPlayerPosition();

}

public void ExitGame()

{

Application.Quit();

}

}

**MapView.cs**

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

using UnityEngine.SceneManagement;

public class MapView : MonoBehaviour

{

public void OpenOptions()

{

SceneManager.LoadSceneAsync("MapViewOptions");

}

public void CharacterMenu()

{

SceneManager.LoadSceneAsync("MapViewOptions");

}

public void InventoryMenu()

{

SceneManager.LoadSceneAsync("MapViewOptions");

}

public void SkillsrMenu()

{

SceneManager.LoadSceneAsync("MapViewOptions");

}

public void ManageMenu()

{

SceneManager.LoadSceneAsync("MapViewOptions");

}

}

**MoneyInput.cs**

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

using TMPro;

public class MoneyInput : MonoBehaviour

{

public TMP\_InputField BronzeText;

public TMP\_InputField SilverText;

public TMP\_InputField GoldText;

public TMP\_InputField PlatinumText;

public int GetInputMoney()

{

int money = 0;

if (int.TryParse(BronzeText.text, out int resultB))

{

money += resultB;

}

if (int.TryParse(SilverText.text, out int resultS))

{

money += resultS \* 100;

}

if (int.TryParse(GoldText.text, out int resultG))

{

money += resultG \* 100 \* 100;

}

if (int.TryParse(PlatinumText.text, out int resultP))

{

money += resultP \* 100 \* 100 \* 100;

}

return money;

}

public void ClearInputs()

{

BronzeText.text = "0";

SilverText.text = "0";

GoldText.text = "0";

PlatinumText.text = "0";

}

}

**PlayerController.cs**

using UnityEngine;

using UnityEngine.Tilemaps;

using System.Collections;

using System.Collections.Generic;

public class PlayerController : MonoBehaviour

{

public Tilemap TileMap;

private bool IsMoving = false;

public bool MoveAfterLoad = false;

public int NightStart = 19;

public int NightEnd = 5;

public GameObject PopUp;

private Dictionary<KeyCode, Vector3Int> DirectionMapping = new Dictionary<KeyCode, Vector3Int>

{

{KeyCode.Keypad7, new Vector3Int(-1, 1, 0)}, // NW

{KeyCode.Keypad8, new Vector3Int(0, 1, 0)}, // N

{KeyCode.UpArrow, new Vector3Int(0, 1, 0)}, // N

{KeyCode.Keypad9, new Vector3Int(1, 1, 0)}, // NE

{KeyCode.Keypad4, new Vector3Int(-1, 0, 0)}, // W

{KeyCode.LeftArrow, new Vector3Int(-1, 0, 0)}, // W

{KeyCode.Keypad6, new Vector3Int(1, 0, 0)}, // E

{KeyCode.RightArrow, new Vector3Int(1, 0, 0)},// E

{KeyCode.Keypad1, new Vector3Int(-1, -1, 0)}, // SW

{KeyCode.Keypad2, new Vector3Int(0, -1, 0)}, // S

{KeyCode.DownArrow, new Vector3Int(0, -1, 0)}, // S

{KeyCode.Keypad3, new Vector3Int(1, -1, 0)} // SE

};

private HashSet<Vector3Int> UniqueDirections = new HashSet<Vector3Int>();

void Update()

{

if (!MoveAfterLoad)

{

transform.position = PlayerManager.Instance.PlayerPosition;

MoveAfterLoad = true;

}

if (!IsMoving && PlayerManager.Instance.CanMove)

{

UniqueDirections.Clear();

foreach (var kvp in DirectionMapping)

{

if (Input.GetKey(kvp.Key))

{

UniqueDirections.Add(kvp.Value);

}

}

if (UniqueDirections.Count > 0)

{

Vector3Int combinedDirection = Vector3Int.zero;

foreach (Vector3Int dir in UniqueDirections)

{

combinedDirection += dir;

}

Vector3Int targetTile = TileMap.WorldToCell(transform.position) + combinedDirection;

if (TileMap.GetTile(targetTile) != null)

{

StartCoroutine(MovePlayerSingle(targetTile));

}

}

}

}

IEnumerator MovePlayerSingle(Vector3Int targetTile)

{

IsMoving = true;

Vector3 targetPosition = TileMap.GetCellCenterWorld(targetTile);

PopUp.SetActive(false);

while (Vector3.Distance(transform.position, targetPosition) > 0.1f)

{

float step = PlayerManager.Instance.GetSpeed() / 4f \* Time.deltaTime;

transform.position = Vector3.MoveTowards(transform.position, targetPosition, step);

yield return null;

}

transform.position = targetPosition;

IsMoving = false;

int tileCost = MainManager.Instance.TileInfoMap[MainManager.Instance.GetTileID(targetTile)].TileMoveCost;

string tileName = MainManager.Instance.TileInfoMap[MainManager.Instance.GetTileID(targetTile)].TileName;

if(tileName == "Road" && PlayerManager.Instance.Skills[1, 4])

{

tileCost = Mathf.RoundToInt(tileCost \* 0.5f);

}

if ((MainManager.Instance.GlobalTime.Hours > NightStart || MainManager.Instance.GlobalTime.Hours < NightEnd) && !PlayerManager.Instance.Skills[1,0])

{

tileCost = Mathf.RoundToInt(tileCost \* 1.5f);

}

float ScoutingMod = 1f - (float)PlayerManager.Instance.Attributes[1] / 100f;

if(ScoutingMod < 0f)

{

ScoutingMod = 0f;

}

PlayerManager.Instance.Stamina -= (float)tileCost \* ScoutingMod;

}

}

**PlayerManager.cs**

using System.Collections.Generic;

using UnityEngine;

using System.IO;

using System.Runtime.Serialization.Formatters.Binary;

using System;

[System.Serializable]

public class PlayerData

{

public long Money;

public Dictionary<int, int> Inventory;

public int MaxStamina;

public float Stamina;

public SerializableVector3 PlayerPosition;

public int PlayerLevel;

public int Speed;

public long XP;

public int Difficulty;

public string Name;

public int SkillPoints;

public int AttrPoints;

public bool[,] Skills;

public int[] Attributes;

public List<Building> Buildings;

public List<Loan> Loans;

public List<Bank> BanksWithFunds;

public List<Quest> Quests;

}

public class PlayerManager : MonoBehaviour

{

public static PlayerManager Instance;

public GameObject player;

public bool CanMove = true;

private PlayerData playerData;

// Saveable

public long Money;

public Dictionary<int, int> Inventory = new Dictionary<int, int>(); // Item ID : Quantity

public int MaxStamina;

public float Stamina;

public Vector3 PlayerPosition;

public int PlayerLevel;

public int Speed;

public long XP;

public int Difficulty;

public string Name;

public int SkillPoints;

public int AttrPoints;

public bool[,] Skills;

public int[] Attributes;

public List<Building> Buildings;

public List<Loan> Loans;

public List<Bank> BanksWithFunds;

public List<Quest> Quests;

public void SavePlayerData(string saveName)

{

BinaryFormatter bf = new BinaryFormatter();

FileStream file = File.Create(Application.persistentDataPath + "/saves/player" + saveName + ".sav");

SerializableVector3 serializablePlayerPosition = new SerializableVector3(PlayerPosition);

PlayerData playerData = new PlayerData

{

Money = Money,

Inventory = Inventory,

MaxStamina = MaxStamina,

Stamina = Stamina,

PlayerPosition = serializablePlayerPosition,

PlayerLevel = PlayerLevel,

Speed = Speed,

XP = XP,

Difficulty = Difficulty,

Name = Name,

SkillPoints = SkillPoints,

AttrPoints = AttrPoints,

Skills = Skills,

Attributes = Attributes,

Buildings = Buildings,

Loans = Loans,

BanksWithFunds = BanksWithFunds,

Quests = Quests,

};

bf.Serialize(file, playerData);

file.Close();

Debug.Log("Saved Player");

}

// Function to load player data from a file

public bool LoadPlayerData(string saveName)

{

string path = Application.persistentDataPath + "/saves/player" + saveName + ".sav";

if (File.Exists(path))

{

BinaryFormatter bf = new BinaryFormatter();

FileStream file = File.Open(Application.persistentDataPath + "/saves/player" + saveName + ".sav", FileMode.Open);

PlayerData playerData = (PlayerData)bf.Deserialize(file);

file.Close();

// Convert PlayerPosition keys back to Vector3Int

Vector3 deserializedPlayerPosition = playerData.PlayerPosition.ToVector3();

Money = playerData.Money;

Inventory = playerData.Inventory;

MaxStamina = playerData.MaxStamina;

Stamina = playerData.Stamina;

PlayerPosition = deserializedPlayerPosition;

PlayerLevel = playerData.PlayerLevel;

Speed = playerData.Speed;

XP = playerData.XP;

Difficulty = playerData.Difficulty;

Name = playerData.Name;

SkillPoints = playerData.SkillPoints;

AttrPoints = playerData.AttrPoints;

Skills = playerData.Skills;

Attributes = playerData.Attributes;

Buildings = playerData.Buildings;

Loans = playerData.Loans;

BanksWithFunds = playerData.BanksWithFunds;

Quests = playerData.Quests;

Debug.Log("Loaded Player");

return true;

}

else

{

Debug.Log("Not Loaded Player");

return false;

}

}

public void InitSkillsAndAttr()

{

MaxStamina = 30;

Stamina = MaxStamina;

Speed = 20;

PlayerLevel = 1;

XP = 0;

Skills = new bool[3, 5];

Attributes = new int[3]{ 1, 1, 1 };

SkillPoints = -(Difficulty - 5);

AttrPoints = SkillPoints \* 3;

switch (Difficulty)

{

case 5:

Money = 10;

break;

case 4:

Money = 50;

break;

case 3:

Money = 100;

break;

case 2:

Money = 500;

break;

case 1:

Money = 1500;

break;

default:

Money = 0; // Default value if y does not match any case

break;

}

}

public void LoadPlayerPosition()

{

if (player != null)

{

player.transform.position = PlayerPosition;

}

}

public int GetSpeed()

{

return Speed + Attributes[1];

}

private void Awake()

{

if (Instance != null)

{

Destroy(gameObject);

return;

}

Instance = this;

DontDestroyOnLoad(gameObject);

}

private void UpdateStamina()

{

MaxStamina = 30;

if ( Stamina > MaxStamina )

{

Stamina = (float)MaxStamina;

}

if (Stamina <= 1 && MaxStamina != 0)

{

Stamina = (float)MaxStamina;

if (MainManager.Instance.GlobalTime.UpdateTime(0, 0, 0, 1, 0)[2] == 1)

{

TickIncome();

}

}

}

public void UpdateLevel()

{

int xt2Nxt = GetXpToNextLevel();

if (XP >= xt2Nxt)

{

XP -= xt2Nxt;

PlayerLevel += 1;

if(PlayerLevel % 5 == 0)

{

SkillPoints += 1;

}

AttrPoints += 1;

}

}

public int GetXpToNextLevel()

{

float baseXP = 10f;

float multiplier = 1.155f;

// 10 + ((x - 1) \* 1.155 ^ x \* log(2, x))

return Mathf.RoundToInt(baseXP + ((PlayerLevel - 1) \* Mathf.Pow(multiplier, PlayerLevel) \* Mathf.Log(PlayerLevel, 2)));

}

public long GetIncome()

{

long income = 0;

foreach( Building build in Buildings )

{

long buildIncome = MainManager.Instance.BuildingTypes[build.BuildingType].Income;

float finalIncomeMod = build.IncomeMod;

if (Skills[2, 4])

{

finalIncomeMod += 0.25f;

}

income += Convert.ToInt64(buildIncome \* finalIncomeMod);

}

foreach (Loan loan in Loans)

{

long payment = Convert.ToInt64((loan.DebtLeft / loan.DurationLeft.ToDays()) \* 1f + loan.InterestToFloat());

income -= payment;

}

return income;

}

public void TickIncome()

{

long income = 0;

foreach (Building build in Buildings)

{

long buildIncome = MainManager.Instance.BuildingTypes[build.BuildingType].Income;

float finalIncomeMod = build.IncomeMod;

if (Skills[2, 4])

{

finalIncomeMod += 0.25f;

}

income += Convert.ToInt64(buildIncome \* finalIncomeMod);

}

foreach (Loan loan in Loans)

{

long payment = Convert.ToInt64((loan.DebtLeft / loan.DurationLeft.ToDays()) \* 1f + loan.InterestToFloat());

Money -= payment;

MainManager.Instance.TileInfoMap[loan.BankTileID].Bank.VaultBalance += payment;

loan.DebtLeft -= Convert.ToInt64((loan.DebtLeft / loan.DurationLeft.ToDays())); ;

loan.DurationLeft.UpdateTime(0, 0, -1, 0);

if (loan.DurationLeft.ZeroTime())

{

Loans.RemoveAll(bankLoan => bankLoan.BankTileID == loan.BankTileID);

}

income -= payment;

}

foreach (Bank bank in BanksWithFunds)

{

bank.TimeTillAccountInterest.UpdateTime(0,0,-1,0);

if (bank.TimeTillAccountInterest.ZeroTime())

{

int tileID = bank.TileID;

int finalAccInt = bank.AccountInterest;

if (Skills[0, 4])

{

finalAccInt += 5;

}

float interest = 1f + (finalAccInt \* 0.01f);

bank.TimeTillAccountInterest.UpdateTime(0, 0, 7, 0);

bank.PlayerBalance = Convert.ToInt64(bank.PlayerBalance \* interest);

}

}

Money += income;

}

private void Update()

{

if (player != null)

{

UpdateStamina();

UpdateLevel();

PlayerPosition = player.transform.position;

}

else

{

player = GameObject.FindGameObjectWithTag("Player");

}

}

}

**QuestMenu.cs**

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

using UnityEngine.UI;

using TMPro;

public class QuestMenu : MonoBehaviour

{

public GameObject listRowPrefab;

public RectTransform contentPanel;

void Start()

{

UpdateList(0);

LayoutRebuilder.ForceRebuildLayoutImmediate(contentPanel);

float contentHeight = contentPanel.rect.height;

contentPanel.anchoredPosition = new Vector2(contentPanel.anchoredPosition.x, -(contentHeight / 2));

}

public void UpdateList(int filter)

{

foreach (Transform child in contentPanel)

{

Destroy(child.gameObject);

}

int index = 0;

foreach (Quest quest in PlayerManager.Instance.Quests)

{

if (filter == 0 || (filter < 4 && filter == quest.Type + 1) || (filter > 3 && (filter - 4) == (quest.Completed ? 1: 0)))

{

GameObject newItem = Instantiate(listRowPrefab, contentPanel);

TextMeshProUGUI[] textComponents = newItem.GetComponentsInChildren<TextMeshProUGUI>();

string color = "FFEECC";

if (quest.Completed)

{

color = "089016";

}

else if (quest.Tracked)

{

color = "FFC800";

}

textComponents[0].text = "<color=#" + color + ">" + quest.Name + "</color>";

textComponents[1].text = "<color=#" + color + ">" + quest.Description + "</color>";

textComponents[2].text = "<color=#" + color + ">" + MainManager.Instance.MoneyToString(quest.Reward) + "</color>";

textComponents[3].text = "<color=#" + color + ">" + quest.TypeString() + "</color>";

textComponents[4].text = "<color=#" + color + ">" + quest.Expires.TimeToString() + "</color>";

GameObject trackBtnObj = newItem.transform.Find("Col 6/TrackBtn").gameObject;

if (quest.Completed)

{

trackBtnObj.SetActive(false);

}

else

{

trackBtnObj.SetActive(true);

Button trackBtn = trackBtnObj.GetComponent<Button>();

TextMeshProUGUI btnText = trackBtn.GetComponentInChildren<TextMeshProUGUI>();

int id = index;

trackBtn.onClick.AddListener(() => ToggleTrackAction(id, filter));

if (quest.Tracked)

{

btnText.text = "<color=#" + color + ">Untrack</color>";

}

else

{

btnText.text = "<color=#" + color + ">Track</color>";

}

}

}

index++;

}

}

public void ToggleTrackAction(int ID, int filter)

{

PlayerManager.Instance.Quests[ID].Tracked = !PlayerManager.Instance.Quests[ID].Tracked;

UpdateList(filter);

}

}

**TileMapChecker.cs**

using UnityEngine;

using UnityEngine.UI;

using UnityEngine.Tilemaps;

using System.Collections.Generic;

using TMPro;

public class TileMapChecker : MonoBehaviour

{

// Collision /w player

public Tilemap Tilemap;

public GameObject PlayerObject;

// City Menu Update

public GameObject CityMenuObj;

public GameObject MenuBG;

public GameObject UIObj;

public CityMenu CityMenu;

// Logic

private bool EnteredCity;

private bool WasSetActive = true;

void Start()

{

if(MainManager.Instance.GameLoaded == false)

{

MainManager.Instance.TileIDMap = new Dictionary<Vector3Int, int>();

MainManager.Instance.TileInfoMap = new Dictionary<int, TileInfo>();

MainManager.Instance.GenerateTileIDs(Tilemap);

}

else

{

PlayerObject.transform.position = PlayerManager.Instance.PlayerPosition;

}

}

private void Update()

{

UpdatePlayerMovementState();;

}

private void UpdatePlayerMovementState()

{

if (CityMenuObj.activeSelf != WasSetActive)

{

PlayerManager.Instance.CanMove = !CityMenuObj.activeSelf;

WasSetActive = CityMenuObj.activeSelf;

}

}

private void OnTriggerStay2D(Collider2D collision)

{

if (collision.gameObject == PlayerObject)

{

Vector3Int playerTilePosition = Tilemap.WorldToCell(PlayerObject.transform.position);

TileBase tile = Tilemap.GetTile(playerTilePosition);

if (tile != null && tile.name.Contains("City"))

{

if (!EnteredCity)

{

int tileID = MainManager.Instance.GetTileID(playerTilePosition);

var tileInfo = MainManager.Instance.TileInfoMap[tileID];

foreach (Quest quest in PlayerManager.Instance.Quests)

{

if (quest.TriggerID == 0)

{

quest.TriggerID = 1;

quest.Description = "Buy 5 Wood and Sell it at another city";

PlayerManager.Instance.Money += quest.Reward;

quest.Reward = 15;

}

}

foreach (var item in tileInfo.TilePrices)

{

for (int i = 0; i < MainManager.Instance.Items.Length; i++)

{

if (MainManager.Instance.Items[i].Name == item.Key)

{

MainManager.Instance.Items[i].LocalPrice = item.Value;

break;

}

}

}

MenuBG.SetActive(true);

CityMenuObj.SetActive(true);

UIObj.SetActive(false);

CityMenu.UpdateCityMenu(tileID);

EnteredCity = true;

}

}

else

{

EnteredCity = false;

}

}

}

}

**UITextUpdater.cs**

using TMPro;

using UnityEngine;

using UnityEngine.UI;

public class TextUpdater : MonoBehaviour

{

public TextMeshProUGUI LevelText;

public TextMeshProUGUI MoneyText;

public TextMeshProUGUI SpeedText;

public Slider StaminaBar;

public Slider XPBar;

public TextMeshProUGUI TimeText;

public GameObject LightTracker;

void Update()

{

// Guild of Old Bankers and Lords of the Income Network

// Grand Order Bank of Loans, Investments and Numbers

LevelText.text = "";

// Update the TextMeshPro text with the variable value

LevelText.text += "<color=#089016>LEVEL</color>\n";

LevelText.text += PlayerManager.Instance.PlayerLevel;

MoneyText.text = "";

// Update the TextMeshPro text with the variable value

MoneyText.text += "<sprite name=\"Money Icon\">" + MainManager.Instance.MoneyToString(PlayerManager.Instance.Money) + "\n";

long income = PlayerManager.Instance.GetIncome();

if (income > 0)

{

MoneyText.text += "<color=#089016>+";

}

else if (income < 0)

{

MoneyText.text += "<color=#C10303>";

}

else

{

MoneyText.text += "<color=#FFEECC>+";

}

MoneyText.text += MainManager.Instance.MoneyToString(income) + "</color>\n";

SpeedText.text = "";

// Update the TextMeshPro text with the variable value

SpeedText.text += "<sprite name=\"Speed Icon\">\n";

SpeedText.text += PlayerManager.Instance.GetSpeed();

StaminaBar.value = (PlayerManager.Instance.Stamina / (float)PlayerManager.Instance.MaxStamina) \* 100;

XPBar.value = ((float)PlayerManager.Instance.XP / (float)PlayerManager.Instance.GetXpToNextLevel()) \* 100;

TimeText.text = "";

// Update the TextMeshPro text with the variable value

TimeText.text += MainManager.Instance.GlobalTime.Hours + ":";

if(MainManager.Instance.GlobalTime.Minutes < 10)

{

TimeText.text += "0";

}

TimeText.text += MainManager.Instance.GlobalTime.Minutes+ "\n";

TimeText.text += "Day: " + (MainManager.Instance.GlobalTime.Days + 1) + "\n";

TimeText.text += "M: " + (MainManager.Instance.GlobalTime.Months + 1) + " Y: " + MainManager.Instance.GlobalTime.Years;

LightTracker.transform.eulerAngles = new Vector3(LightTracker.transform.eulerAngles.x, LightTracker.transform.eulerAngles.y, (MainManager.Instance.GlobalTime.Hours - 1) \* (-15f));

}

}