

Requirements and Analysis Document for Pawntastic

Jacob Bredin, Martin Jonsson,
Jonathan Lindqvist, Mathias Prétot

October 24, 2021

Contents

1	Introduction	1
1.1	Definitions, acronyms, and abbreviations	1
2	Requirements	2
2.1	User Stories	2
2.1.1	Basic World #7 - <i>Implemented</i>	2
2.1.2	Pawns #9 - <i>Implemented</i>	2
2.1.3	Structures #8 - <i>Implemented</i>	3
2.1.4	Pawn Role Management #16 - <i>Implemented</i>	3
2.1.5	Richer World #11 - <i>Implemented</i>	3
2.1.6	Pawn Pathfinding #13 - <i>Implemented</i>	4
2.1.7	Resource Collection #31 - <i>Implemented</i>	4
2.1.8	Colony Shared Inventory #10 - <i>Implemented</i>	4
2.1.9	Build Using Resources #12 - <i>Implemented</i>	5
2.1.10	Structures spawn pawns #72 - <i>Implemented</i>	5
2.1.11	Pawn Starvation #15 - <i>Implemented</i>	6
2.1.12	Pawn Inventory and Stockpiles #18 - <i>Implemented</i>	6
2.1.13	Persistence #29 - <i>Implemented</i>	6
2.1.14	Role Stations #70	7
2.1.15	Main Menu #30	7
2.1.16	Passive Animals #45	8
2.1.17	Incident System #46	8
2.1.18	Raiders #47	8
2.1.19	Colony Walls #49	9
2.1.20	Pawn Collision #17	9
2.2	Definition of Done	9
2.3	User interface	10
2.3.1	Start Menu	10
2.3.2	New Game	10
2.3.3	Load	11
2.3.4	Settings	12
2.3.5	Game View	13
2.3.6	Tiles	14
2.3.7	Beings	14
2.3.8	Pawn-panel	15
2.3.9	Toolbar	15
2.3.10	Event log	16
2.3.11	Game menu	16
2.3.12	Save	17

3	Domain model	18
3.1	Class responsibilities	19
3.1.1	Tiles	19
3.1.2	BeingGroups and Beings	19
4	External Dependencies and Tools	20
4.1	Tools	20
4.2	Application Dependencies	21
4.3	Testing Dependencies	21
	References	22

1 Introduction

Pawntastic is a game where a player is in control of a medieval-themed colony, and the goal is to survive for as long as possible. Threats such as natural disasters, raids from other groups of beings and starvation are always present. The game takes full advantage of object-oriented design. The current scope of the game allows the player to play locally with and against a simulation, the player is in control of building construction and role designation. The target audience is gamers that seek a niche genre that satisfies the need for simplicity and relaxation whilst also having a lurking threat to be wary of.

1.1 Definitions, acronyms, and abbreviations

Expansion: A set of User Stories that when completed marks a milestone.

Being: A singular independent entity in the world.

Pawn: A human-like Being who is a member of the Colony.

Being Group: A group of Beings that belong and act together.

Player: The user who is playing the game.

Colony: A Being Group of Pawns managed by the Player.

Action: A set of steps executed by a Being to perform a simple task.

Role: A giver of Actions in a specific order to fulfill a certain responsibility.

Tile: A section of the world containing a single object.

Terrain: A part of the underlying World that has specific characteristics, like how easily it can be traversed.

Structure: A building belonging to a Colony, existing in the World.

World: The environment the game takes place in.

Resource: A natural resource located in the in the World.

Item: Collected from a Resource to be used for constructing Structures.

Inventory: A collection of Items.

HP: Health Points, health for either structure (structural integrity) or beings.

UI: User Interface, the part of the program that is responsible for communicating with the player.

NPC: Non Player Character, a Being that is not owned by nor managed by the player.

Javadoc: A documentation tool for Java.

CPD: Copy/Paste Detector, a tool from PMD that checks if there is any copied and pasted code for the project.

SOLID: The SOLID principles, single responsibility principle, open-closed principle, Liskov substitution principle, interface segregation principle, and dependency inversion principle.

MVC: Model View Controller, a software design pattern for separating the applications business logic (model) from the presentation logic (view) and user-input logic (controller).

2 Requirements

2.1 User Stories

All user stories planned for the game. Implemented User Stories have the *Implemented*-tag in header.

2.1.1 Basic World #7 - *Implemented*

Description

As a player I need to see the game world to be able to play the game.

Confirmation

Functional

- Generate empty world.
- Display the world.

2.1.2 Pawns #9 - *Implemented*

Description

As a player I need to see my pawns so that I know what they are doing.

Confirmation

Functional

- Pawns are part of a colony.

- Pawns move somewhere directly to prove that they can move (doesn't matter where or when).
- Pawns can be seen by the player.

2.1.3 Structures #8 - *Implemented*

Description

As a player, I want to be able to place structures in order to expand the colony.

Confirmation

Functional

- Structures are place-able.
- Structures can be seen by the player.

2.1.4 Pawn Role Management #16 - *Implemented*

Description

As a player I want to be able to assign my pawns to specific roles in order to have greater control over my colony.

Confirmation

Functional

- There exists different roles: lumberjack, farmer, guard, miner, fisher, builder.
- There is a mechanism for assigning idle pawns to roles when requested.
- There is a menu for assigning a number of pawns to each role. The number can not exceed the total number of pawns.
- A pawn can switch roles when the player sets a quota for a role.

2.1.5 Richer World #11 - *Implemented*

Description

As a player I need my pawns to have access to resources from the world.

Confirmation

Functional

- The world must contain natural resources: trees, stones, water.

- World generation should be coherent, not random.

2.1.6 Pawn Pathfinding #13 - *Implemented*

Description

As a player I want my pawns to intelligently move to the places they need to because I don't want to wait.

Confirmation

Functional

- Pawns no longer randomly pick destinations in the world.
- Pawns know where they should go.
- Pawns can generate a path to their destination.
- Pawns follow the path.
- Obstacle avoidance.

2.1.7 Resource Collection #31 - *Implemented*

Description

As a player, I want my pawns to be able to collect the resources in the world, in order to amass resources.

Confirmation

Functional

- Roles create actions.
- Beings perform actions.
- The pawns must go up to resources to collect them (without destroying them).
- The pawns needs to find the nearest resource.
- A pawn only collects resources from what their role is assigned.

2.1.8 Colony Shared Inventory #10 - *Implemented*

Description

As a player I need the colony to have an inventory that can have items added and removed, and I need to be able to see them.

Confirmation

Functional

- The colony has a shared inventory.
- The player can see which items are contained in the inventory.
- Pawns can access it to store or retrieve items when needed.

2.1.9 Build Using Resources #12 - *Implemented*

Description

As a player I want my pawns to consume resources when building structures, to make the game more challenging and realistic.

Confirmation

Functional

- All newly placed buildings start at 0% built.
- Unbuilt structures need to have resources delivered to them by pawns.
- Built percentage is based on how many materials are in the materials inventory.
- Once all resources are delivered the blueprint turns into a building.
- Materials can be removed from a building making it unbuilt.

2.1.10 Structures spawn pawns #72 - *Implemented*

Description

As a player I want pawns to spawn when structures are fully constructed, because I want more pawns to manage.

Confirmation

Functional

- When a structure is fully built pawns spawn next to the structure.
- The first structure to place is a town hall.
- The town hall spawns the first pawns in the colony.

2.1.11 Pawn Starvation #15 - *Implemented*

Description

As a player I want to have to fight off starvation because I want a challenge.

Confirmation

Functional

- Pawns have hunger and health.
- Once hunger reaches 0, pawns start to take damage.
- When a pawn's health reaches 0, they die.
- When a pawn's hunger reaches 50% they eat from their inventory.
- A pawn prioritizes hunger over their role work.

2.1.12 Pawn Inventory and Stockpiles #18 - *Implemented*

Description

As a player I want my pawns to carry resources and have stockpiles to store resources in, because I want my pawns to have to go to stockpiles when fetching resources.

Confirmation

Functional

- Pawns have an inventory with a finite carrying capacity that can contain some resources.
- Items have weight.
- Stockpile buildings can be built and displayed.
- Stockpiles have infinite inventories and all share the same inventory.
- When a pawn does not have a resource it needs in its inventory, it goes to a stockpile to fetch it.
- When gathering resources, they are deposited into the inventory of the pawn.

2.1.13 Persistence #29 - *Implemented*

Description

As a player, I want to be able to save the state of my game, so I can return to it at a later date.

Confirmation

Functional

- Being able to save the current state of the game.
- Save on quit (no choice of save file).
- Load on start (no choice of save file).

2.1.14 Role Stations #70

Description

As a user, I want my pawns to go to their role-specific stations to acquire tools, in order for them to perform their role.

Confirmation

Functional

- Each role has a role station.
- There is a menu for selecting which station to build.
- Before performing the main actions of a role, the action of acquiring the tools is performed.

2.1.15 Main Menu #30

Description

As a player, I want there to be a main menu where I can 1) start new game, 2) pick which save file to load and 3) quit the game.

Confirmation

Functional

- When the program is started, the main menu is visible and not the game.
- There is a button for starting a new game.
- There is a button for picking which save file to load.
- There is a button for quitting the game.

2.1.16 Passive Animals #45

Description

As a player, I want wildlife in the world to make the game feel more alive.

Confirmation

Functional

- There are sheep in the world.
- Sheep look different from Pawns.
- The sheep randomly roam around the world (maybe in herds).
- Wool can be harvested from the sheep by a shepherd.

2.1.17 Incident System #46

Description

As a player, I want incidents to occur during gameplay in order to spice things up a bit.

Confirmation

Functional

- There is a framework for random generation of incidents.
- Incidents occur at dynamic probabilities.
- here is some kind of assurance that incidents will occur at reasonable frequency.
- An event can create or spawn things in the World.

2.1.18 Raiders #47

Description

As a player, I want there to be raids so that I can defend against them.

Confirmation

Functional

- There is a system for letting Beings know about each other so they can attack each other.
- There is a Raider that attacks Pawns.
- Pawns with the Guard role attack Raiders.

- Raiders attack structures.

2.1.19 Colony Walls #49

Description

As a player, I want there to be colony walls that block enemies from entering my colony, so I can protect my Pawns.

Confirmation

Functional

- The walls are unwalkable.
- There are gates that only Pawns and Passive Animals can walk through.
- Gates have less HP and can be broken by Enemies.

2.1.20 Pawn Collision #17

Description

As a player I do not want my pawns to occupy the same space because I want to see all my pawns.

Confirmation

Functional

- Two pawns can't be in the same location at the same time.
- Pawns avoid each other when pathfinding.

2.2 Definition of Done

For a User Story to be considered done it needs to satisfy the following criteria:

- All Tasks in the User Story have been completed.
- All Tasks should have tests and pass them with at least 90% overall coverage.
- A UML class diagram of the newly implemented user story should be done and included in the `readme.md` -file of the parent package.
- The code for the User Story has to be reviewed and approved by at least two other group members.

2.3 User interface

2.3.1 Start Menu

The start menu, see figure 1, is the first view the player is greeted with.

- “New game” takes the player to a new window where the player gets to set up the world.
- “Quick start” starts the game from the most recent save-file.
- “Load game” opens up the load game window.
- “Settings” opens the settings window.
- “Exit game” quits the game.



Figure 1: A sketch of the start-menu of *Pawntastic*.

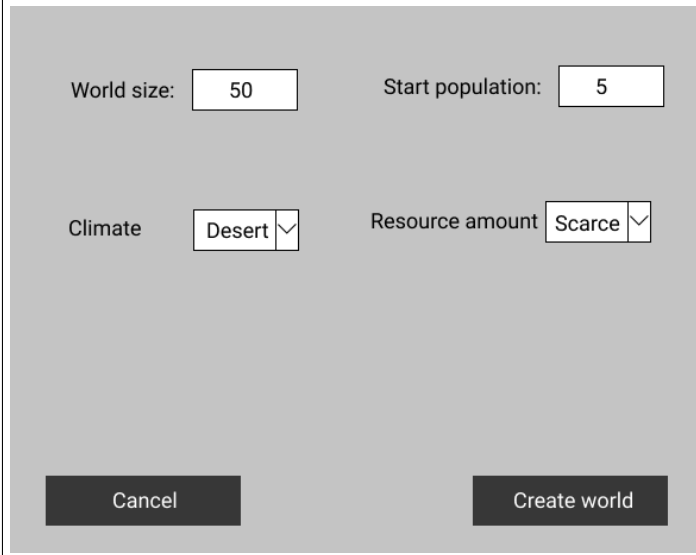
2.3.2 New Game

Here, the player gets to set up the world with the following settings, see figure 2:

- World Size: the number of tiles the world width and height should be.
- Start Population: the number of pawns the colony should start with.
- Climate: what kind of climate the world should have for example desert, mountain, marsh, et cetera.

- Resource amount: decides how resource-rich the world will be, for example scarce, plentiful or normal.

The player can go back to the start menu by pressing the “Cancel” button or press “Create World” to start the game.



World size: 50 Start population: 5

Climate: Desert ▼ Resource amount: Scarce ▼

Cancel Create world

Figure 2: A sketch of the menu for creating the world.

2.3.3 Load

Here, the player can select which file to load and then press the “Load Game” button to load it, see figure 3. Pressing the “Cancel” button will return the player to the previous view.

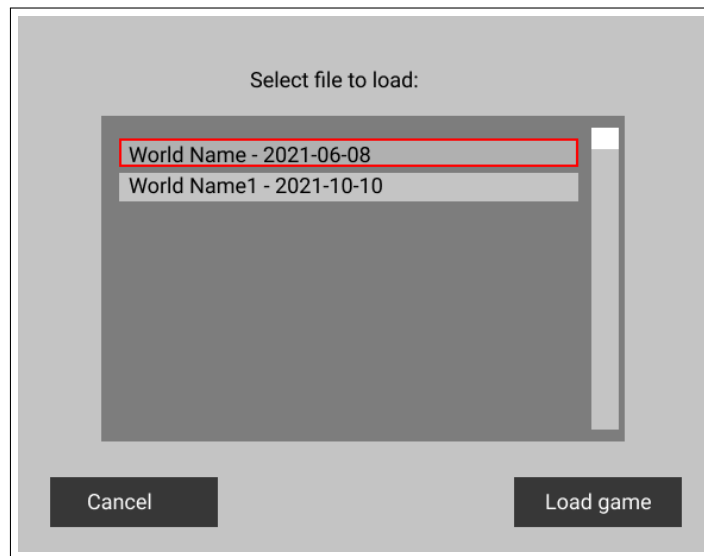


Figure 3: A sketch of the load window used to select load file.

2.3.4 Settings

Here, the player can adjust the following settings, see figure 4:

- Display mode: whether the game is full-screen or windowed.
- Resolution: what resolution the game should render at, for example 1920x1080, 1600x900 et cetera.
- Vertical sync: whether or not the frame rate should be limited to the refresh rate of the screen.
- Sound: a volume slider used to adjust the sound level of the game.

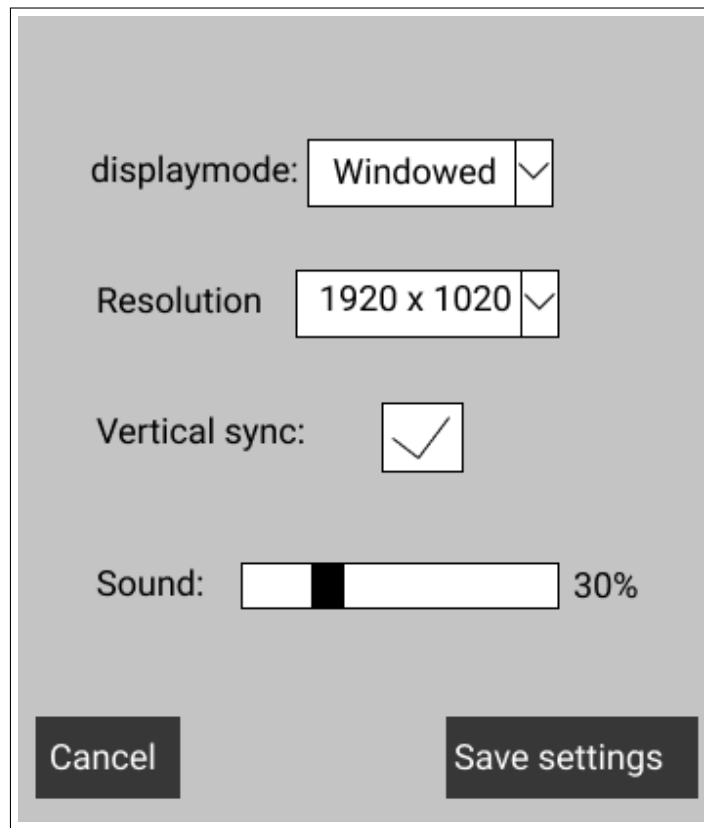


Figure 4: A sketch of the settings menu.

2.3.5 Game View

The game view, see figure 5, consists of 3 panels: the pawn-panel, the toolbar and the game screen.

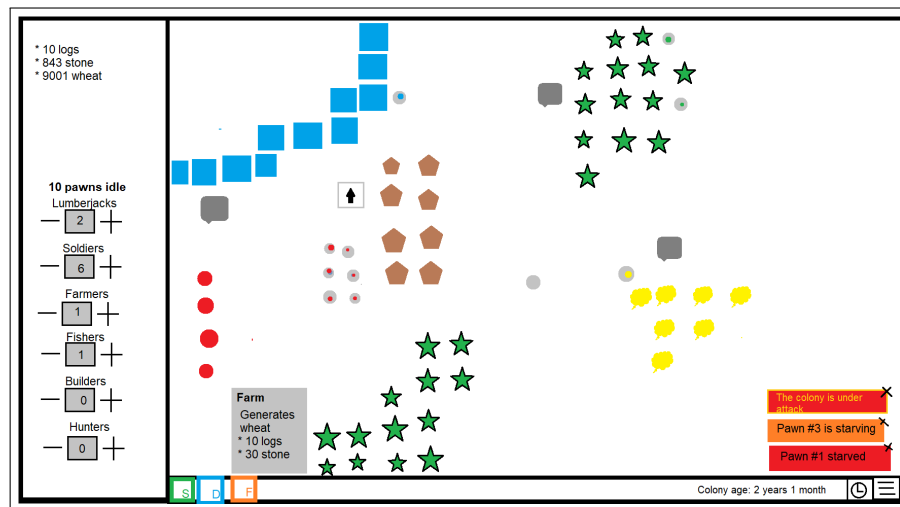


Figure 5: A sketch showing an overview of the planned final game interface.

2.3.6 Tiles

The game world itself consists of different kinds of Tiles, as shown in figure 5:

- The blue rectangles are water Tiles.
- The green stars are trees.
- The brown pentagons are houses.
- The yellow blobs are wheat fields.
- The grey rounded rectangles are rocks.

2.3.7 Beings

Besides Tiles, the World will have Beings moving inside of it, as shown in figure 5. Grey circles are Pawns belonging to the Colony, and the smaller circle inside of the grey one shows what Role it has been assigned to. For example, if the Pawn has a green circle then they are a lumberjack and can only chop wood, while a blue one is a fisher that can fish in the water, et cetera.

Red circles are enemies of the Colony, and the red Pawns are guards that will defend the Colony against these enemies.

2.3.8 Pawn-panel

Using the Role-allocation panel, see figure 6, the player can allocate how many beings in the Colony should be assigned to each Role type. The player can only allocate idle Pawns which means if there are no idle Pawns then the player needs to deallocate some Pawns before giving them new Roles.

On this panel the player can also see what Resources the Colony has gathered.

* 10 logs
* 843 stone
* 9001 wheat

10 pawns idle

Lumberjacks	—	2	+
Soldiers	—	6	+
Farmers	—	1	+
Fishers	—	1	+
Builders	—	0	+
Hunters	—	0	+

Figure 6: A sketch of the panel used to allocate Pawn Roles.

2.3.9 Toolbar

In order to build a Structure, the player need to select what kind of Structure to build and select a Tile where the Structure should be placed. For example, if the Player wants to order the construction of a farm, they can do it either by pressing the F-key on the keyboard or clicking the orange button on the toolbar, see figure 7, and then press an empty Tile.

On the right end of the toolbar there are two buttons, the first being the clock which the player can press to open an event log, see figure 8. The second button is the menu button which opens the menu window, see figure 9.



Figure 7: A sketch of the toolbar used to select Structures to place and access other menus.

2.3.10 Event log

A small view where the Player can scroll through events that have occurred, see figure 8. Events can for example be that a structure has been completed or an enemy raid has started or ended. The event log can be closed by pressing the “X” button.

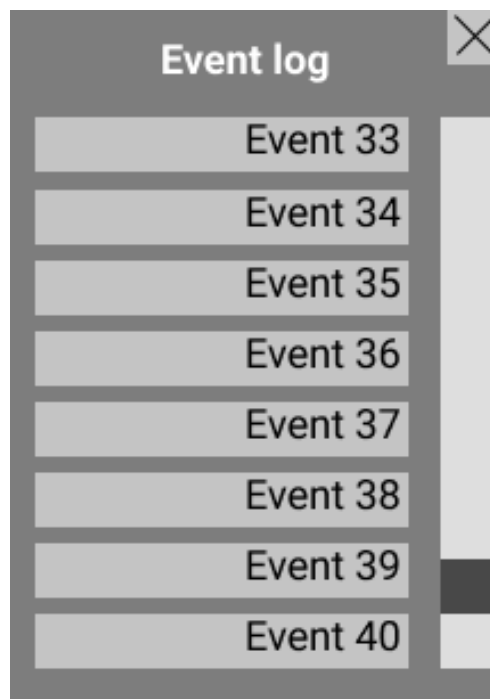


Figure 8: A sketch of the event log, showing all recent events.

2.3.11 Game menu

The game menu, see figure 9, gives almost the same options as the start menu. The differences being:

- “Save” opens the save window, see figure 10.
- “Quick save” Saves the game in a new file.
- “Quit to start menu” returns the Player to the start menu. Will warn the Player if the game has not recently been saved.

- “Return to game” resumes the game.

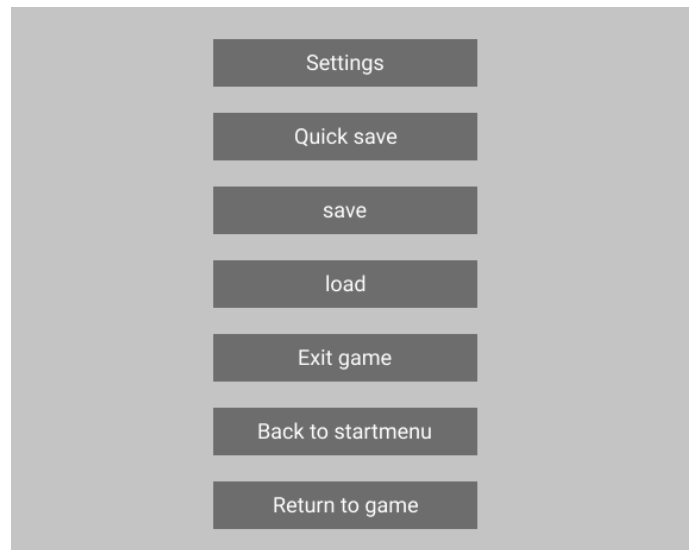


Figure 9: The game menu which gives the player access to most other menus.

2.3.12 Save

Here, the player is given two options to save: either overwrite an existing save-file, or create a new one, see figure 10. The “Save Game” button saves the game according to what the player has selected. If nothing has been selected, then this button is disabled.

The “Cancel” button aborts the process and returns the player to the game.

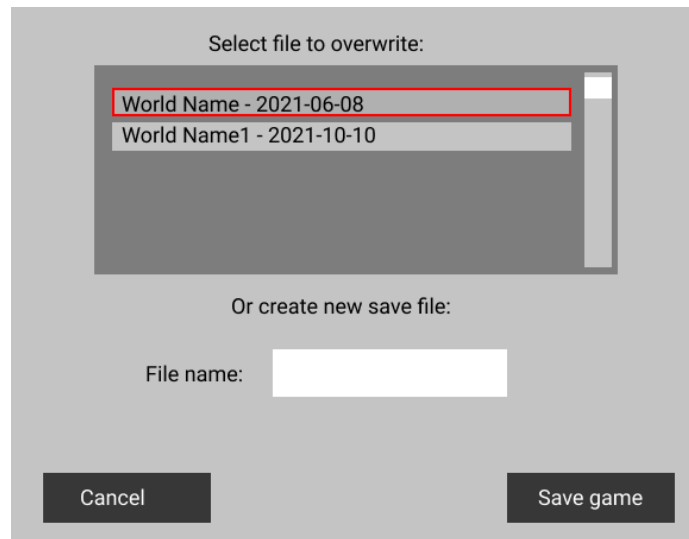


Figure 10: A sketch of the save window.

3 Domain model

The domain model of the game, see figure 11, is the most abstract view of which parts make up the game.

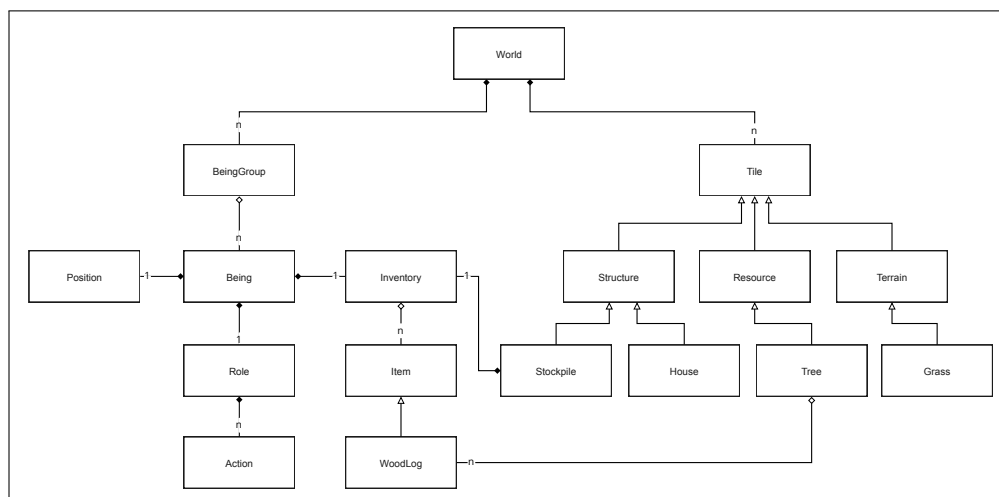


Figure 11: The complete domain model of the game.

3.1 Class responsibilities

The `BeingGroup`s and `Tile`s in the game belong to a `World` that is responsible for updating them and keeping track of them.

3.1.1 Tiles

The `Tile`s make up the environment that the game exists in, and that environment is laid out as a grid of squares, with each square being a single `Tile`. They are responsible for knowing what exists in their space and its properties.

There are three types of `Tile`:

1. `Structure`
2. `Resource`
3. `Terrain`

`Structure`s are buildings created by pawns after a player has ordered their construction. A single `Structure` is responsible for its unique functionality and state. `House` is a simple `Structure`, and is a place for `Beings` to rest. A more complex `Structure` is `Stockpile` which is responsible for storage of `Item`s using its internal `Inventory`.

`Resource` is a type of tile that is randomly spread out throughout the world, and is responsible for giving out a type of material. From the `Tree`, a type of `Item` can be harvested: `WoodLog`.

`Terrain` represents the underlying environment type of that tile in the world. It is responsible for the movement speed of `Being`s that try to move on top of it. For example, when moving on `Grass` they move at full speed, but on some other terrain they might move slower.

3.1.2 BeingGroups and Beings

In the `World`, there are multiple `BeingGroup`s, and one of them is controlled by the player. Each `BeingGroup` is an aggregation of one or more `Being`s. They are responsible for keeping track of their constituents and assigning them `Role`s.

A `Being` is responsible for keeping track of its own location, inventory and role using `Position`, `Inventory` and `Role`, respectively. Depending on

their assigned `Role`, they can act entirely different. For example, a `Being` can act as:

- a Pawn, by having its `Role` assignment controlled by the Player.
- an Enemy, by having its `Role` be a raider.
- a passive animal, by having its `Role` be a roaming role.

Pawns are what the `Being`s controlled by the player are called. They belong to a `BeingGroup` known as the colony. The colony is unique in that there exists only one and it is the only group to be influenced by the actions of the player. Even though Pawns are a part of the player controlled colony, they can not be controlled directly.

A `Role` is responsible for handing out `Action`s that when performed in succession result in execution of a specific job.

4 External Dependencies and Tools

The game is built using multiple external libraries and tools. Here, you will find all of them listed.

4.1 Tools

- Git [1] is used for source control.
- GitHub [2] is used as a remote repository and SCRUM board.
- Travis CI [3] is used for continuous integration.
- GitHub Actions [4] is also used for continuous integration.
- Miro [5] is used for creation of UML diagrams.
- Apache Maven [6] is used as a build pipeline.
- Apache Maven Checkstyle Plugin [7] is used to force adherence to a strict code style.
- Apache Maven Compiler Plugin [8] is used to compile the program.
- Apache Maven Shade Plugin [9] is used to package the application along with all its dependencies in a JAR file.

4.2 Application Dependencies

- OpenJDK 15 [10] is used to compile and run the game.
- libGDX [11] is used as the game engine.
- Guava [12] is used as a utilities library.

4.3 Testing Dependencies

- JUnit [13] is used as a unit test framework and test runner.
- AssertJ [14] is used to make more fluent assertions in unit tests.
- Mockito [15] is used to create mocked instances of dependencies in unit tests.

References

- [1] *Git*, ver. 2.33.0, Aug. 2021. [Online]. Available: <https://git-scm.com/>, Accessed on 09/29/2021.
- [2] Microsoft, *GitHub*. [Online]. Available: <https://github.com/>, Accessed on 09/29/2021.
- [3] *Travis CI*. [Online]. Available: <https://www.travis-ci.com/>, Accessed on 09/29/2021.
- [4] —, *GitHub Actions*. [Online]. Available: <https://docs.github.com/en/actions>, Accessed on 10/07/2021.
- [5] *Miro*, 2021. [Online]. Available: <https://miro.com/>, Accessed on 09/29/2021.
- [6] The Apache Software Foundation, *Maven*, ver. 3.8.2, Aug. 2021. [Online]. Available: <https://maven.apache.org/>, Accessed on 09/29/2021.
- [7] —, *Apache Maven Checkstyle Plugin*, ver. 3.1.2, Jan. 2021. [Online]. Available: <http://maven.apache.org/plugins/maven-checkstyle-plugin/>, Accessed on 09/29/2021.
- [8] —, *Apache Maven Compiler Plugin*, ver. 3.8.1, Apr. 2019. [Online]. Available: <https://maven.apache.org/plugins/maven-compiler-plugin/>, Accessed on 09/29/2021.
- [9] —, *Apache Maven Shade Plugin*, ver. 3.2.4, May 2020. [Online]. Available: <https://maven.apache.org/plugins/maven-shade-plugin/>, Accessed on 09/29/2021.
- [10] Oracle, *OpenJDK*, ver. 15, Sep. 2020. [Online]. Available: <http://openjdk.java.net/projects/jdk/15/>, Accessed on 09/29/2021.
- [11] *libGDX*, ver. 1.10.0, Apr. 2021. [Online]. Available: <https://libgdx.com/>, Accessed on 09/29/2021.
- [12] Google, *Guava*, ver. 31.0-jre, Sep. 2021. [Online]. Available: <https://guava.dev/>, Accessed on 09/29/2021.
- [13] J. Team, *JUnit*, ver. 5.7.0, Sep. 2020. [Online]. Available: <https://junit.org/junit5/>, Accessed on 09/29/2021.
- [14] *AssertJ - fluent assertions java library*, ver. 3.20.2, Jun. 2021. [Online]. Available: <https://assertj.github.io/doc/>, Accessed on 09/29/2021.
- [15] *Mockito framework*, ver. 3.12.4, Aug. 2021. [Online]. Available: <https://site.mockito.org/>, Accessed on 09/29/2021.