Columbus Project : New Heaven

Created by

Suttanop Chanah 6632239821

Akarachai Yongsuwankul 6630357821

2110215 Programming Methodology

Semester 1 Year 2024
Chulalongkorn University

Columbus Project : New Heaven

After a catastrophic nuclear war rendered Earth uninhabitable, humanity embarked on the Columbus Project, a desperate mission to find a new home among the stars. After a thousand years of searching, a planet was discovered—New Heaven, a lush world offering hope for survival. But humanity's remnants, divided into rival factions, now compete fiercely to claim the planet. Who will prevail and rebuild civilization, and who will be lost to history? The battle for New Heaven begins.

Introduction

Columbus Project: New Heavens is inspired by the strategic board game "Catan" and designed as a 2-player strategy game built using Java and JavaFX. Set in a post-apocalyptic world, this game challenges players to build and defend colonies on a new planet, "New Heaven." By collecting resources, constructing buildings, and attacking opponents, players aim to reduce their rival's colony HP to zero, ensuring their survival.

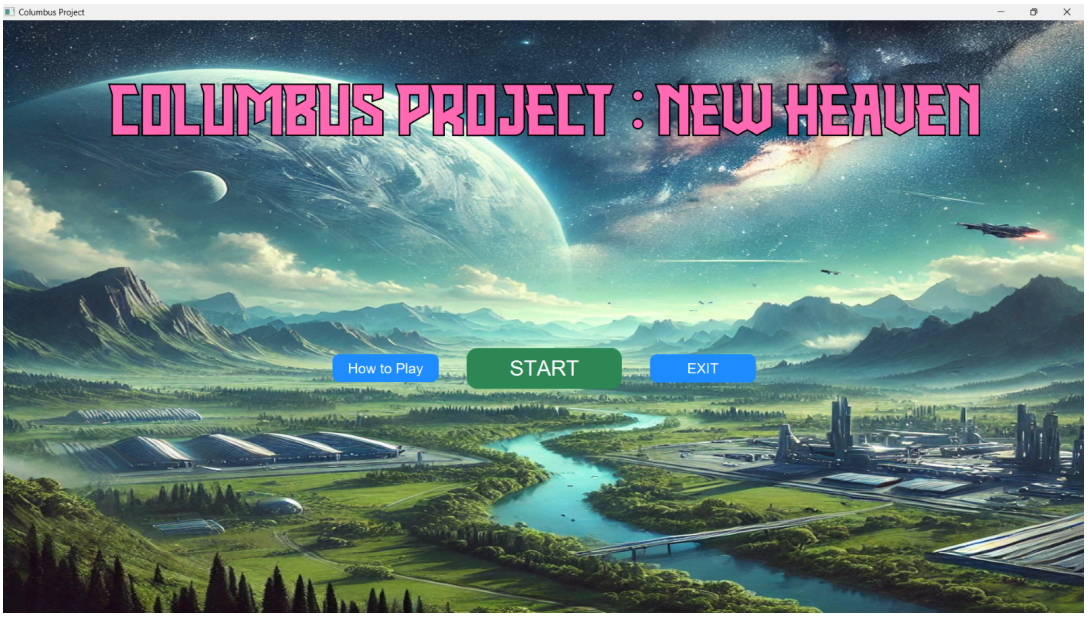
The gameplay incorporates elements of luck, strategy, and resource management. Players roll dice to acquire resources distributed across hexagonal tiles. These resources are then utilized to perform actions such as building, producing resources, attacking, and upgrading structures.

Game Rules

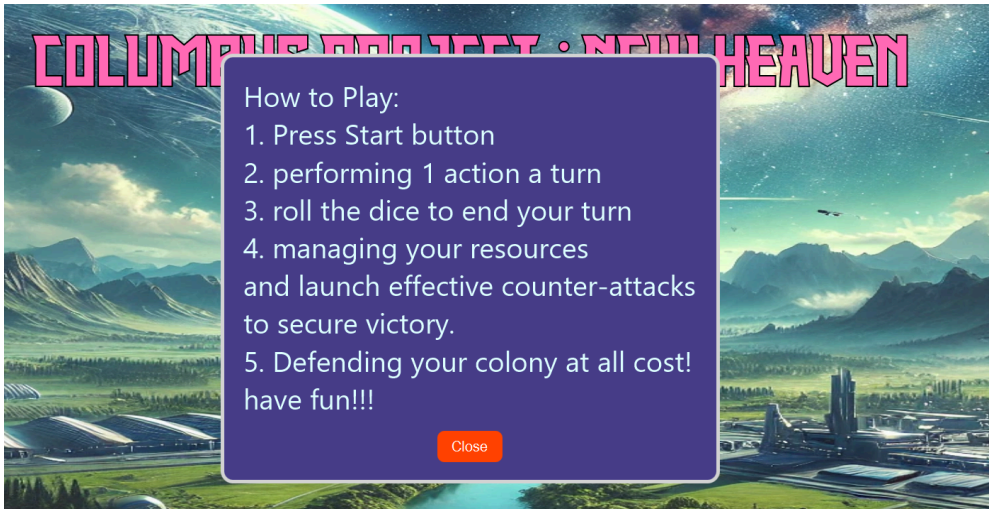
- Each player starts with a Colony (20 Hp) placed on opposite ends of the hexagonal map. Colonies represent the player's base and must be defended at all costs.
- Players take turns performing 1 of 4 actions:
  - Build/Upgrade : Construct or upgrade new buildings using collected resources, players can only build adjacent to their buildings.
    - Quarry: Hp: 3, Produce: 1 cost: 3 Copper
    - Factory: Upgrade of quarry, Hp: 4, Produce: 3 cost: 4 Oil
    - Military Camp: Hp: 5, Atk: 2, cost : 3 Vibranium can attack only adjacent hex

        - Missile Fortress: Upgrade of military camp, Hp:8, Atk: 3, cost: 8 Uranium can attack all over the map.
    - Produce: Generate resources using specialized buildings (e.g., Factories or Quarries).
    - Attack: Use attackable buildings to damage the opponent's building.
    - Alchemize: exchange amount of resources with another resource. Alchemize rate of copper : oil : vibranium : uranium : jojolium is 1 : 2 : 4 : 6 : 8 (if has remainder, round up)
- Players roll dice at the end of their turn to generate resources from the tiles matching the dice values.
- The game ends when one player's colony's HP reaches zero.

MainMenu scene



How To Play scene



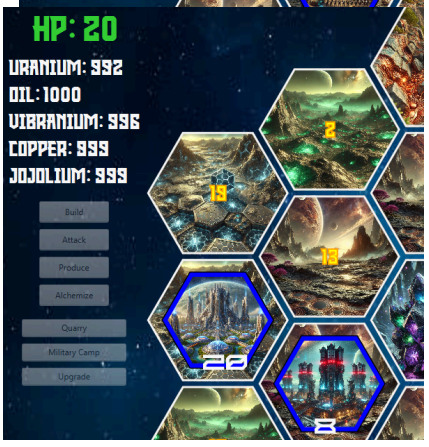
Build Action

highlighted tiles that can be able to construct building

You will begin with your Colony located at the leftmost tiles and rightmost tile of the map for player 1 and player 2 respectively. When you click the **Build** button, the system will highlight the tiles where you can either build or upgrade. These tiles are those adjacent to your existing buildings.

To build, you can select a tile to build or upgrade from the highlighted tiles. After selecting a tile, choose the building you want to construct or upgrade. The building options will only be enabled if you have enough resources to construct that specific building. Once the construction is successful, the selected tile will display the new building with hp indicated below, and the resources required for the construction will be deducted from your inventory.

Build on adjacent tile



Upgrade an upgradable building

Attack Action

Attack Mechanism Explanation

1. **Initiating an Attack:**
- Select a building capable of attacking, such as a **Military Camp** or **Missile Fortress**.
- Once the attack action is triggered, the game will highlight targetable tiles. These tiles represent adjacent or reachable buildings owned by the opponent.

There are 2 buildings that player 1 can be selected for attack

2. **Choosing a Target:**
- From the highlighted tiles, click on the target building you want to attack. Note that each attackable building has a different range of attack.

Missile Fortress can select any opponent's building due to its attack range

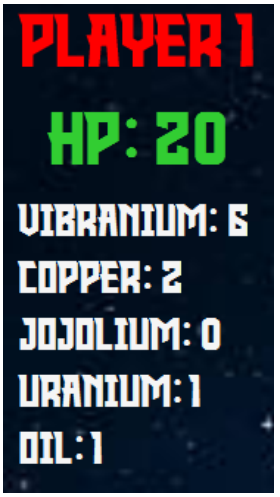
3. **Executing the Attack:**
- The attacking building will deal damage based on its attack power.
- If the target's health reaches zero, the building is destroyed, and the tile becomes empty.
- If the target survives, its health will be reduced accordingly, allowing for future attacks.

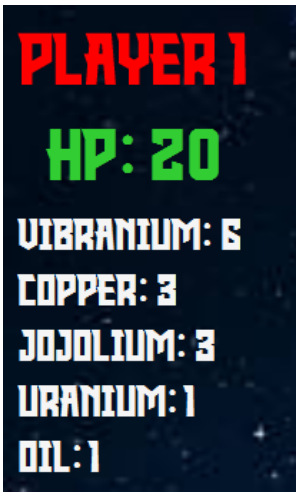
The selected target will take damage equal to the attack value of the building used for the attack.

4. **Action Limitations:**
- Each building can perform only one attack per turn.
- Plan your strategy carefully to maximize the impact of your attacks while protecting your own buildings.

Produce Action

On board, player 1 has a quarry on copper hex and a factory on jojolium hex. If player 1 produces a resource, player will gain 1 copper and 3 jojolium.

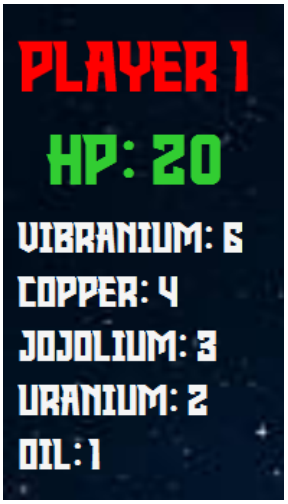


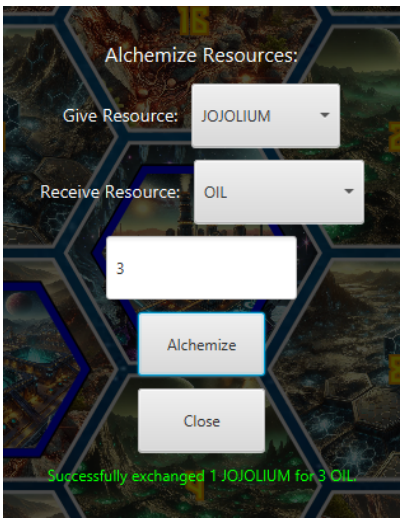


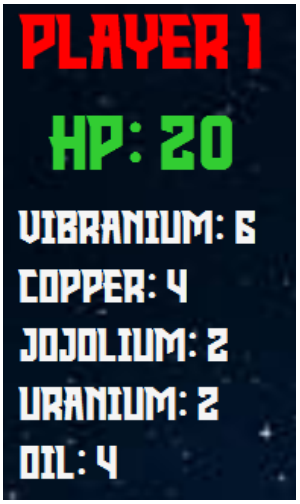
before after

Alchemize Action

In the game, If a player wants to build a factory, the player must have 4 oil, so the player can alchemize 1 Jojolium to 3 oil.







before alchemize after

Victory: End Game Triggered

The end game is triggered under the following condition:

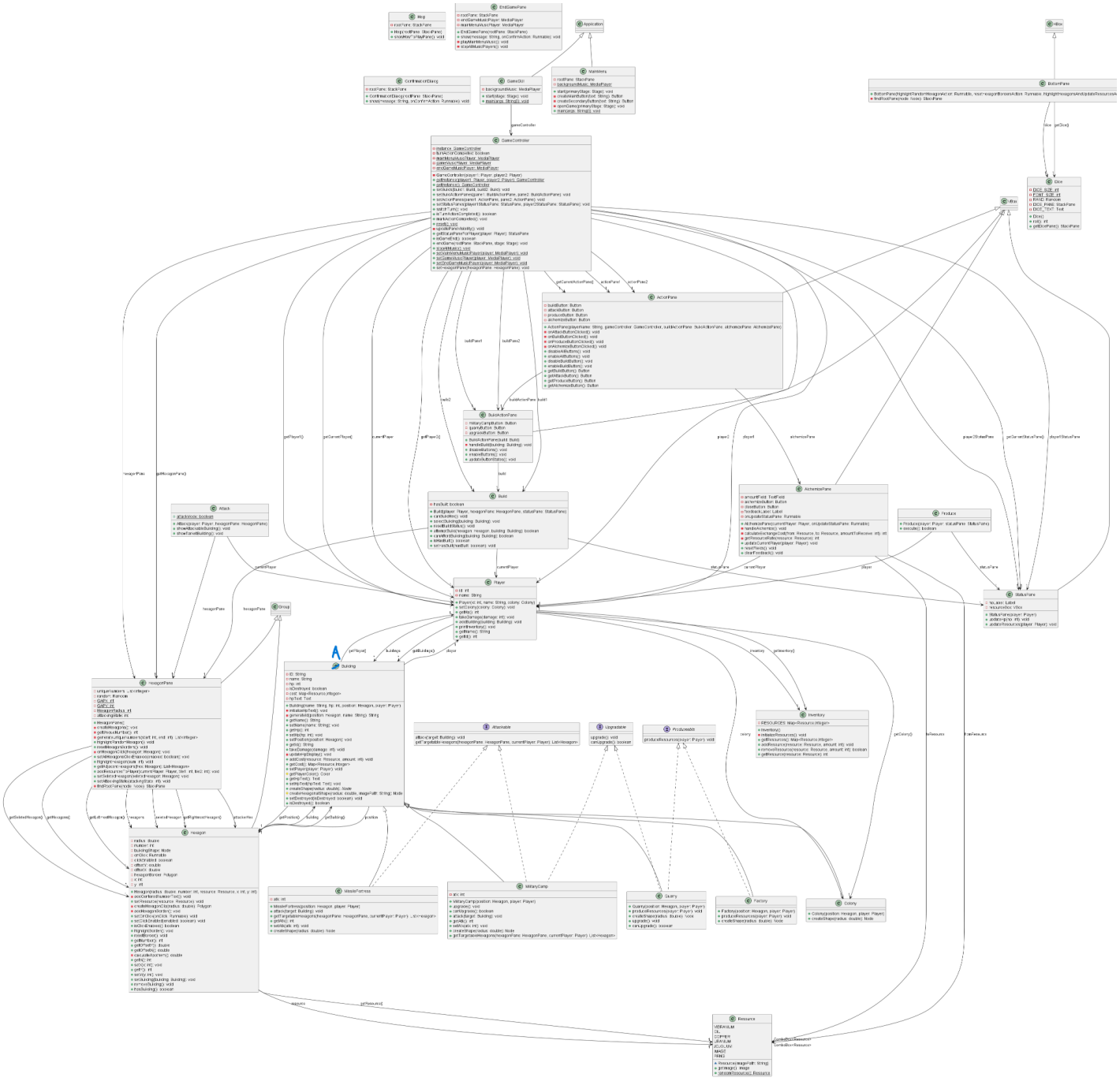
1. Player Health Reaches Zero:

- If the health points (HP) of either player (Player 1 or Player 2) are reduced to 0 or below, the game will immediately end.
- The player whose HP remains above 0 is declared the winner.

Example:

In this game, now is player 1 turn, player 2 has only 2 hp, so player 1 decides to use missile fortress to attack the colony.Then, player 1 wins.

Class diagram (Link diagram for classification below)



***You can refer to the classification diagram at the following link:

https://drive.google.com/file/d/1b1yTz6QGrz4VPb2tE1EvHuMJ3QY4wE3X/view?usp=sharing

1. Package board

1.1 Hexagon

1.1.1 Fields

- double radius	Size of the hexagon.
- int number	The unique number assigned to the hexagon.
- Resource resource	The resource type associated with the hexagon.
- Building building	Associated building on the hexagon.
- Node buildingShape	The visual representation of the building.
- Runnable OnClick	Action triggered when the hexagon is clicked.
- boolean clickEnabled	Indicates if the hexagon can be clicked.
- double offsetY	Offset for positioning the hexagon in Y-axis.
- double offsetX	Offset for positioning the hexagon in X-axis.
- int x	X-coordinate of the hexagon.
- int y	Y-coordinate of the hexagon.

- Polygon hexagonBorder	The graphical border of the hexagon.

1.1.2 Constructor

+ Hexagon(double radius, int number, Resource resource, int x, int y)	Initializes a hexagon with specified radius, resource, and coordinates.

1.1.3 Method

- void addCenteredNumberText()	Return this tile to its normal appearance.
- Polygon createHexagonClip(double radius)	Creates a polygon clip for the hexagon shape.
- void addHexagonBorder()	Adds a border to the hexagon.
- double calculateApothem()	Calculates the apothem of the hexagon.
+ void setResource(Resource resource)	Sets the resource image of the hexagon.
+ void setOnClick(Runnable onClick)	Sets the action to trigger on hexagon click.
+ void setClickEnabled(boolean enabled)	Enables or disables clicking on the hexagon.

+ boolean isClickEnabled()	Checks if clicking is enabled.
+ void highlightBorder()	Highlights the border of the hexagon.
+ void resetBorder()	Resets the border to default color.
+ void setBuilding(Building building)	Places a building on the hexagon.
+ void removeBuilding()	Removes the building from the hexagon.
+ boolean hasBuilding()	Checks if the hexagon has a building.
Generate getter/setter of all fields	

1.2 Enum Resources

1.2.1 Enum

The enum defines the following resource types:

VIBRANIUM("/images/vibranium.png"),

OIL("/images/oil.png"),

COPPER("/images/copper.png"),

URANIUM("/images/uranium.png"),

JOJOLIUM("/images/jojolium.png");

1.2.2 Fields

- final Image IMAGE	Instance field to store the image
- final static Random PRNG	Random instance for randomization.

1.2.3 Constructor

Resource(String imagePath)	initializes the image for each resource

1.2.3 Methods

+ static Resource randomResource()	Static method to get a random resource
Getter for IMAGE	

2. Package building

2.1 abstract Building

2.1.1 Fields

- final String ID	Identifier based on hash of tile number and type
- final String NAME	Name of the building
- int hp	Hit Points of the building
- Hexagon position	Hexagon where the building is constructed
- boolean isDestroyed	Indicates if the building has been destroyed
- Player player	Player who owns the building.
- Map<Resource,Integer> cost	A map of resources and their required amounts for constructing
- Text hpText	Text to display Building HP

2.1.2 Constructor

+ Building(String name, int hp, int prosperityPoints, Hexagon position, Player player)	Initializes the building with its name, HP, prosperity points, position, owner, and other necessary attributes.

2.1.3 Methods

- void initializeHpText()	Creates a Text object to display the building's current HP
- String generateId(Hexagon position, String name)	Generate a hash ID based on the tile number and building name
- void updateHpDisplay()	Updates the Text object displaying the building's current HP.
+ void takeDamage(int damage)	Reduces the building's HP by the specified amount. Marks the building as destroyed if HP reaches 0. Notifies the GameController if the building is a Colony.
+ void addCost(Resource resource, int amount)	Adds a resource and its required amount to the building's construction or upgrade cost.
+ abstract Node createShape(double radius)	Abstract method for defining the visual representation of the building.
# Color getPlayerColor()	Returns the color associated with the owning player
# Node createHexagonalShape(double radius, String imagePath)	Creates a hexagonal shape for the building with an image pattern and player-specific color. Also includes the Text object displaying the building's HP.
Getter/Setter for all fields	

2.2 Class Colony entends Building

2.2.1 Constructor

+ Colony(Hexagon position, Player player)	Initializes colony for each player, this also represents player’s HP.

2.2.2 Methods

+ createShape(double radius)	Creates the visual representation of the Colony using the createHexagonalShape method from the Building class. It applies the colony's image (/images/colony.png) to the hexagonal shape.

2.3 Class Factory entends Building implements Produceable

2.3.1 Constructor

+ Factory(Hexagon position, Player player)	Initializes a Factory with a predefined name ("Factory"), HP (10), position, and owning player. Adds an initial construction cost requiring 4 units of OIL.

2.3.2 Methods

+ produceResources(Player player)	Implements the Produceable interface. Produces 3 units of the resource associated with the Factory's position and adds them to the owning player's inventory.
+ createShape(double radius)	Creates the visual representation of the Factory using the createHexagonalShape method from the Building class. It applies the factory's image (/images/factory.png) to the hexagonal shape.

2.4 Class MilitaryCamp entends Building implements Attackable,Upgradable

2.4.1 Fields

- int atk	attack power of this building

2.4.2 Constructor

+ MilitaryCamp(Hexagon position, Player player)	Initializes a MilitaryCamp with a predefined name ("MilitaryCamp"), HP (3), position, and owning player. Sets an initial attack power

	(3) and adds a construction cost requiring 3 units of VIBRANIUM.

2.4.3 Methods

+ void upgrade()	Replaces the MilitaryCamp with a MissileFortress on the same position and owned by the same player.
+ boolean canUpgrade()	Checks if the player has sufficient resources to upgrade to a MissileFortress.
+ attack(Building target)	Implements the Attackable interface. Attacks the target building, dealing damage equal to the camp's attack power. If the target is destroyed, it is removed from the game board.
+ createShape(double radius)	Creates the visual representation of the MilitaryCamp using the createHexagonalShape method from the Building class. It applies the building's image (/images/MilitaryCamp.png) to the hexagonal shape.

+ List<Hexagon> getTargetableHexagons(HexagonPane hexagonPane, Player currentPlayer)	Returns a list of adjacent hexagons containing enemy buildings.
Getter/setter for all fields	

2.5 Class MissileFortress entends Building implements Attackable

2.5.1 Fields

- int atk	attack power of this building

2.5.2 Constructor

+ MissileFortress(Hexagon position, Player player)	Initializes a MilitaryCamp with a predefined name ("MilitaryCamp"), HP (3), position, and owning player. Sets an initial attack power (3) and adds a construction cost requiring 3 units of VIBRANIUM.

2.5.3 Methods

+ attack(Building target)	Implements the Attackable interface. Attacks the target building, dealing damage equal to the camp's attack power. If the target is destroyed, it is removed from the game board.
+ createShape(double radius)	Creates the visual representation of the MilitaryCamp using the createHexagonalShape method from the Building class. It applies the building's image (/images/missile.png) to the hexagonal shape.
+ List<Hexagon> getTargetableHexagons(HexagonPane hexagonPane, Player currentPlayer)	Returns a list of all hexagons containing enemy buildings, enabling a global attack range.
Getter/setter for all fields	

2.6 Class Quarry entends Building implements Produceable,Upgradable

2.6.1 Constructor

+ Quarry(Hexagon position, Player player)	Initializes a Quarry with a predefined name ("Quarry"), HP (3), position, and owning player. Adds a construction cost requiring 3 units of COPPER.

2.6.2 Methods

+ produceResources(Player player)	Implements the Produceable interface. Produces 1 unit of the resource associated with the Quarry's position and adds it to the owning player's inventory. Handles errors if the position is null.
+ createShape(double radius)	Creates the visual representation of the Quarry using the createHexagonalShape method from the Building class. It applies the quarry's image (/images/quarry.png) to the hexagonal shape.
+ void upgrade()	Transforms the current Quarry building into a Factory. Updates the Hexagon position's building.
+ boolean canUpgrade()	Checks if the player has sufficient resources to upgrade the Quarry to a Factory.

3. Package building.interfaces

3.1 interface Attackable

3.1.1 Methods

void attack(Building target)	Defines the action of attacking a target building. Implementations must specify the logic for reducing the target's HP or handling its destruction.

3.2 interface Produceable

3.2.1 Methods

void produceResources(Player player)	Defines the action of producing resources and adding them to the specified player's inventory. Implementations must specify the logic for determining the type and amount of resources produced.

3.3 interface Upgradable

3.3.1 Methods

void upgrade()	Performs the upgrade action, transforming the current instance into its upgraded version.
boolean canUpgrade()	Checks if the upgrade conditions are met, such as resource availability.

4. Package game

4.1 Class Attack

4.1.1 Fields

- Player currentPlayer	Represents the player whose turn it is to attack.
- HexagonPane hexagonPane	Manages the hexagonal game board and facilitates interactions like highlighting hexagons and setting states.
+ static boolean attackMode	Static field that indicates whether the game is currently in attack mode. Default as false

4.1.2 Constructor

+ Attack(Player player, HexagonPane hexagonPane)	Initializes the Attack class with the current player and the game board (HexagonPane).

4.1.3 Methods

+ void showAttackableBuilding()	Identifies and highlights all buildings owned by the current player that can attack (i.e., implement Attackable). Enables clicking only on these buildings and sets attackMode to true.

+ void showTarketBuilding()	Highlights valid target buildings based on the selected attackable building's type (MilitaryCamp or MissileFortress). Enables clicking on valid targets for the attack.

4.2 Class Build

4.2.1 Fields

- Player currentPlayer	The player currently performing the building action.
- HexagonPane hexagonPane	The pane containing the game hexagons.
- StatusPane statusPane	The pane displaying the player's resources and status.
- boolean hasBuilt	Indicates whether the player has built during the current turn.

4.2.2 Constructor

+ Build(Player player, HexagonPane hexagonPane,StatusPane statusPane)	Constructor to initialize the Build object with the current player, game hexagons, and status pane.

4.2.3 Methoods

+ void canBuildHex()	Highlights adjacent hexagons where the current player can construct or upgrade buildings.
+ void selectBuilding(Building building)	Attempts to select and build the specified building on the selected hexagon. Logs a message if no building is selected.
+ void resetBuildStatus()	Resets the build status for the player, enabling building in a new turn.
+ boolean attemptBuild(Hexagon hexagon, Building building)	Attempts to construct a building on the specified hexagon. Deducts resources if successful and prevents further building during the turn.
+ boolean canAffordBuilding(Building building)	Checks if the player has enough resources to construct the specified building.
Getter/Setter for hasBuilt	

4.2 Class Dice

4.2.1 Fields

- static final int DICE_SIZE	Constant defining the size of the dice face in pixels. DICE_SIZE = 100
- static final int FONT_SIZE	Constant defining the font size for the text displaying the dice value.FONT_SIZE = 50
- final Random RAND	A Random instance for generating random dice rolls.
- final StackPane DICE_PANE	A container for the graphical representation of the dice, including the face and value text.
- final Text DICE_TEXT	A Text node for displaying the current value of the dice.

4.2.2 Constructor

+ Dice()	Initializes the dice with a graphical representation, including a square face with rounded corners and a centered text value set to "1".

4.2.3 Methods

+ int roll()	Method to roll the dice and update the displayed value
Getter for DicePane	

4.3 Class GameController

4.3.1 Fields

- static GameController instance	Singleton instance of the GameController.
- Player player1	Represents Player 1.
- Player player2	Represents Player 2.
- Player currentPlayer	The player currently taking their turn.
- HexagonPane hexagonPane	The game board containing all hexagonal tiles.
- BuildActionPane buildPane1	The build action pane for Player 1.
- BuildActionPane buildPane2	The build action pane for Player 2.
- Build build1	Build logic for Player 1.
- Build build2	Build logic for Player 2.
- StatusPane player1StatusPane	The status pane for Player 1.
- StatusPane player2StatusPane	The status pane for Player 2.

- ActionPane actionPane1	The action pane for Player 1.
- ActionPane actionPane2	The action pane for Player 2.
- boolean turnActionCompleted	Flag indicating if an action has been completed in the current turn.

4.3.2 Contructor

- GameController(Player player1, Player player2)	Initializes the GameController with Player 1, Player 2, and the game board (HexagonPane). Defaults the current player to Player 1.

4.3.3 Methods

+ static GameController getInstance(Player, Player)	Returns the singleton instance of GameController. Initializes it if it doesn't exist.
+ static GameController getInstance()	Retrieves the existing GameController instance.
+ void switchTurn()	Switches the current player, resets turn flags, updates pane visibility, and enables new turn actions.
+ void markActionCompleted()	Marks the current turn's action as completed and disables buttons for further actions.
+ static void reset()	Resets the singleton instance to null.

+ void resetHexagonBorders()	Resets all hexagon borders, disables clicking, and clears the selected hexagon and attack state.
+ boolean isGameEnd()	Checks if the game has ended based on players' health points (HP).
+ void endGame(StackPane rootPane,Stage stage)	Displays the end-game pane and redirects to the main menu.
+ static void stopAllMusic()	Centralized method to stop all music
- void updatePaneVisibility()	Updates the visibility of action and builds panes based on the current player's turn.
Getter/Setter for all fields	

4.4 Class Produce

4.4.1 Fields

- Player player	The player for whom resources are being produced.
- StatusPane statusPane	The status pane associated with the player, used to visually update the player's resource inventory.

4.4.2 Contructor

+ Produce(Player player, StatusPane statusPane)	Initializes the Produce instance with a player and their associated status pane.

4.4.3 Methods

+ boolean execute()	Iterates through the player's buildings, produces resources from Produceable buildings, and updates the StatusPane.

5. Package player

5.1 Class Player

5.1.1 Fields

- final int ID	Unique identifier for the player (e.g., 1 or 2).
- final String NAME	The player's name.
- final Inventory INVENTORY	The player's inventory, which tracks resources.
- Colony colony	The colony owned by the player.
- List<Building> buildings	A list of buildings owned by the player.

5.1.2 Constructor

+ Player(int id, String name,Colony colony)	Initializes the player with a unique ID, name, colony, and an empty inventory and building list.

5.1.3 Methods

+ void addBuilding(Building building)	Adds a new building to the player's list and logs the addition.
+ void printInventory()	Logs the player’s inventory, listing resources and their respective quantities. THIS METHOD IS FOR DEBUGGING ONLY
Getter/Setter for all fields	

5.2 Class Inventory

5.2.1 Fields

- final Map<Resource, Integer> RESOURCES	A map tracking the quantities of each resource.

5.2.2 Constructor

+ Inventory()	Initializes the inventory with all resources set to an initial amount.

5.2.3 Methods

- void initializeResources()	Initialize all resources
+ void addResource(Resource resource, int amount)	Increases the quantity of the specified resource by the given amount.

+ boolean removeResource(Resource resource, int amount)	Decreases the quantity of the specified resource by the given amount if sufficient quantity exists. Returns true if successful, false otherwise.
+ int getResource(Resource resource)	Retrieves the current quantity of the specified resource.
Getter/Setter for all fields	

6. Package view

6.1 Class MainMenu extends Application

6.1.1 Fields

- StackPane rootPane	Root layout pane for managing multiple layers.
- MediaPlayer backgroundMusic	MediaPlayer for background music

6.1.2 Methods

+ void start(Stage primaryStage)	Sets up the main menu, including the background, title, and buttons.
- Button createMainButton(String text)	Creates a styled primary button (e.g., for "Start").

- Button createSecondaryButton(String text)	Creates a styled secondary button (e.g., for "How to Play" and "Exit").
- void openGame(Stage primaryStage)	Launches the main game interface by transitioning to the GameGUI.

6.2 Class GameGUI extends Application

6.2.1 Fields

- GameController gameController	Centralized controller for managing the game state and flow.
- MediaPlayer backgroundMusic	MediaPlayer for background music

6.2.2 Methods

+ void start(Stage stage)	Initializes the game window, GUI components, and game elements like players, colonies, and panes.
+ static main(String[] args)	Launches the game application.

6.3 Class Help

6.3.1 Fields

- StackPane rootPane	Reference to the root StackPane, which serves as the base for the entire GUI.

6.2.2 Constructor

+ Help(StackPane rootPane)	Constructor to initialize the Help object with the root StackPane reference.

6.2.3 Methods

+ void showHowToPlayPane()	Displays the "How to Play" instructions in a new overlay pane.

7. Package pane

7.1 Class ActionPane extends VBox

7.1.1 Fields

- Button buildButton	Button of build action.
- Button attackButton	Button of attack action.
- Button produceButton	Button of produce action.
- Button alchemizeButton	Button of alchemize action.
- BuildActionPane buildActionPane	Pane for select building.

- AlchemizePane alchemizePane	Pane for alchemize .

7.1.2 Constructor

+ ActionPane(String playerName, GameController gameController, BuildActionPane buildActionPane, AlchemizePane alchemizePane)	Constructor to initialize the pane for select action. - set padding to 10 - set alignment to top center - create all action button - set all on click action with method below - set all button with width = 100 height = 30. - add all button to this - set this translateY to 30

7.1.3 Methods

- void onAttackButtonClicked()	Set buildActionPane visible to false and call attack.showAttackableBuilding()
- void onBuildButtonClicked()	Set attacking state to 0 toggle visibility of BuildActionPane call build.canBuildHex()
- void onProduceButtonClicked()	Set attacking state to 0 resetHexagonBorders call produce.execute() then markActionCompleted()
- void onAlchemizeButtonClicked()	Set attacking state to 0 resetHexagonBorders toggle visibility of AlchemizePane

+ void disableAllButtons()	Set all button disable
+ void enableAllButtons()	Set all button enable
+ void disableBuildButton()	Set BuildButton disable
+ void enableBuildButton()	Set BuildButton enable
+ getter of all button	

7.2 Class AlchemizePane extends VBox

7.2.1 Fields

- ComboBox<Resource> fromResource	Drop down list box for select give resource.
- ComboBox<Resource> toResource	Drop down list box for select receive resource.
- TextField amountField	Field to receive the amount of resource that player wants.
- Button alchemizeButton	Button to confirm alchemize.
- Button closeButton	Button to close alchemize pane
- Label feedbackLebel	Lebel that give feedback to player

- Player currentPlayer	Player that play in this turn
- Runnable onUpdateStatusPane	Action to update status pane.

7.2.2 Constructor

+ AlchemizePane(Player currentPlayer, Runnable onUpdateStatusPane)	Constructor to initialize the pane for alchemize action. - set padding to 10 - set alignment to center - create all label, combobox - group label, combobox with Hbox - create all textfield. button, feedback label - add all children to this - set visible to false

7.2.3 Methods

+ void handleAlchemize()	Collect all player info and remove and add resource that player exchange
- int calculateExchangeCost(Resource from, Resource to, int amountToReceive)	Calculate resource with rate
- int getResourceRate(Resource resource)	Collect resource exchange rate
+ void updateCurrentPlayer(Player player)	Reset fields whenever the current player is updated

+ void resetFields()	Reset all field

7.3 Class BottomPane extends HBox

7.3.1 Fields

- final Dice DICE	Dice that player roll when end turn

7.3.2 Constructor

+ BottomPane(Runnable highlightRandomHexagonAction, Runnable resetHexagonBordersAction, BiConsumer<Integer, Integer> highlightHexagonsAndUpdateResourcesAction, Runnable openMainMenuAction)	Constructor to initialize the pane at bottom - set padding to 20 - set alignment to bottom center - create dice with Vbox to center left - create roll button to action roll - reset all board to change turn and give player resource form dice result - create back to main menu button - set dice to left, back button to right

7.3.3 Methods

- StackPane findRootPane(Node node)	Find the root pane of this

+ getter of dice	

7.4 Class BuildActionPane extends VBox

7.4.1 Fields

- Button militaryCampButton	Button to select military camp building
- Button quarryButton	Button to select quarry building
- Button upgradeButton	Button to upgrade building
- Build build	Build action

7.4.2 Constructor

+ BuildActionPane(Build build)	Constructor to initialize the pane for select building or upgrade - set padding to 20 - set alignment to center - create button for upgradable building and upgrade button - set width to 150, visible false - set on action to button

7.4.3 Methods

- handleBuild(Building building)	if select hex is null and build building, if select hex is upgradeable, upgrade building
+ void disableButtons()	Set all button disable true
+ void enableButtons()	Set all button disable false
+ void updateButtonStates()	Disable all button if already build this turn

7.5 class ConfirmationDialog

7.5.1 Fields

- StackPane rootPane	Reference to the root StackPane

7.5.2 Constructor

+ ConfirmationDialog(StackPane rootPane)	Constructor to initialize the pane for confirm back to menu

7.5.3 Methods

+ void show(String message, Runnable onConfirmAction)	Create pop up pane that have confirm dialog, yes button, no button and set action to button

7.6 Class EndGamePane

7.6.1 Fields

- StackPane rootPane	Reference to the root StackPane
- MediaPlayer endGameMusicPlayer	Music when game end
- MediaPlayer mainMenuMusicPlayer	Music in main menu

7.6.2 Constructor

+ EndGamePane(StackPane rootPane)	Constructor to initialize the pane for end game pop up

7.6.3 Methods

+ show(String message, Runnable onConfirmAction)	Create pop up pane that have end game dialog, back button and set action to button

- void playMainMenuMusic()	Play music when back to main menu
- stopAllMusicPlayers()	Stop all music

7.7 class HexagonPane extends Group

7.7.1 Fields

# List<Hexagon> hexagons = new ArrayList<>();	Hexagon for build board
- List<Integer> uniqueNumbers	Number for each hexagon
- Random random = new Random()	Random number and resource
- int GAPX = 10	Set gap for x axis to 10
- int GAPY = 5	Set gap for y axis to 5
- int HexagonRadius = 100	Set hexagon radius to 15

- Hexagon seletedHexagon	Hexagon that player select
- Hexagon attackerHex	Hexagon that player select to attack
- int attackingState = 0	Set state for attack to 0

7.7.2 Constructor

+ HexagonPane()	Call createHexagons()

7.7.3 Methods

+ void createHexagons()	Create hexagon with random number and resource and sort it to board
- List<Integer> generateUniqueNumbers(int start, int end)	Generate number to each hexagon
+ void highlightRandomHexagon()	Highlight border to random hexagon
+ void resetHexagonBorders()	Reset border of all hexagon

- void onHexagonClick(Hexagon hexagon)	Set selectedHexagon to hexagon that player click
+ void setAllHexagonsClickEnabled(boolean enabled)	Set all hexagon click to enable
+ void highlightHexagon(int sum)	Highlight border of hexagon
+ Hexagon getLeftmostHexagon()	Get most left hexagon
+ Hexagon getRightmostHexagon()	Get most right hexagon
+ List<Hexagon> getAdjacentHexagons(Hexagon hex)	Get list of adjacent hexagon
+ void addResourcesToPlayer(Player currentPlayer, int tile1, int tile2)	Add resource of hexagon that is result of dice to player
+ getter/setter of some field	SeletedHexagon, AttackingState, Hexagons

7.8 class StatusPane extends VBox

7.8.1 Fields

- Label hpLabel	Label that show player hp
- VBox resourceBox	List of label that show player resource

7.7.2 Constructor

+ StatusPane(Player player)	Constructor to initialize the pane for player status - set padding to 10 - set alignment to top center - create hpLabel and resourceBox

7.7.3 Methods

+ void updateHp(int hp)	Set hp text to current player hp
- void updateResources(Player player)	Update resource text to current player inventory