## Part 3

Using polymorphism,inheritance and abstraction

To use the strategy pattern with the Ai's, we made an abstract parent class called "AI". All the Ai's are children of "AI" and they override all the functions from it. When a player is created, we can input a parameter that is of class "AI", therefore we can input any of the Ai's since they are children of "AI". We set a attribute of type "AI" to this parameter and when the player has to do a certain action, let's say conquer, the attribute calls the function "aiConquers" from the appropriate Ai using polymorphism and the overriding mechanism.

We applied inheritance, abstraction and polymorphism in the Observer and Decorator pattern part of the project. The Observer class is abstract and is extended by views and a GameStatsInterface (which is used for decorators). GameStatsInterface is also abstract and adds further methods which must be implemented in its children. GameStatsInterface is extended by GameStats, a basic concrete implementation of GameStatsInterface, and GameStatsDecorator. GameStatsDecorator is further extended by DominationDecorator and VictoryCoinDecorator, allowing more specific information to be displayed to the user.

When using the decorators, the thing we are decorating is a pointer to type GameStatsInterface that is however pointing to an instance of GameStats. When we add decorators, we are then initializing decorators but still using the GameStatsInterface pointer.

One place where we should have used inheritance was for PowerBadgeDeck and RaceBannerDeck. Both of these are very similar and instead of being 2 separate classes, should have both extended a Deck class that had all the functionality that both used (such as draw for example)

For polymorphism, we have certain methods that have different parameters for different use cases. For example our game loop, PlayGame. If we want to allow the user to enter a number of players, it calls the method startGame() that takes in no parameters. Otherwise, like for the automated tournaments, there is a method startGame(int nbPlayers) that takes in a number which represents the number of players and automatically starts the game.

High cohesion and low coupling

To promote high cohesion and low coupling, we separated the ".cpp" files in folders that are linked with each other.

Furthermore, there is only one instance of a map and it is a global variable so it is easy to access from any other class.

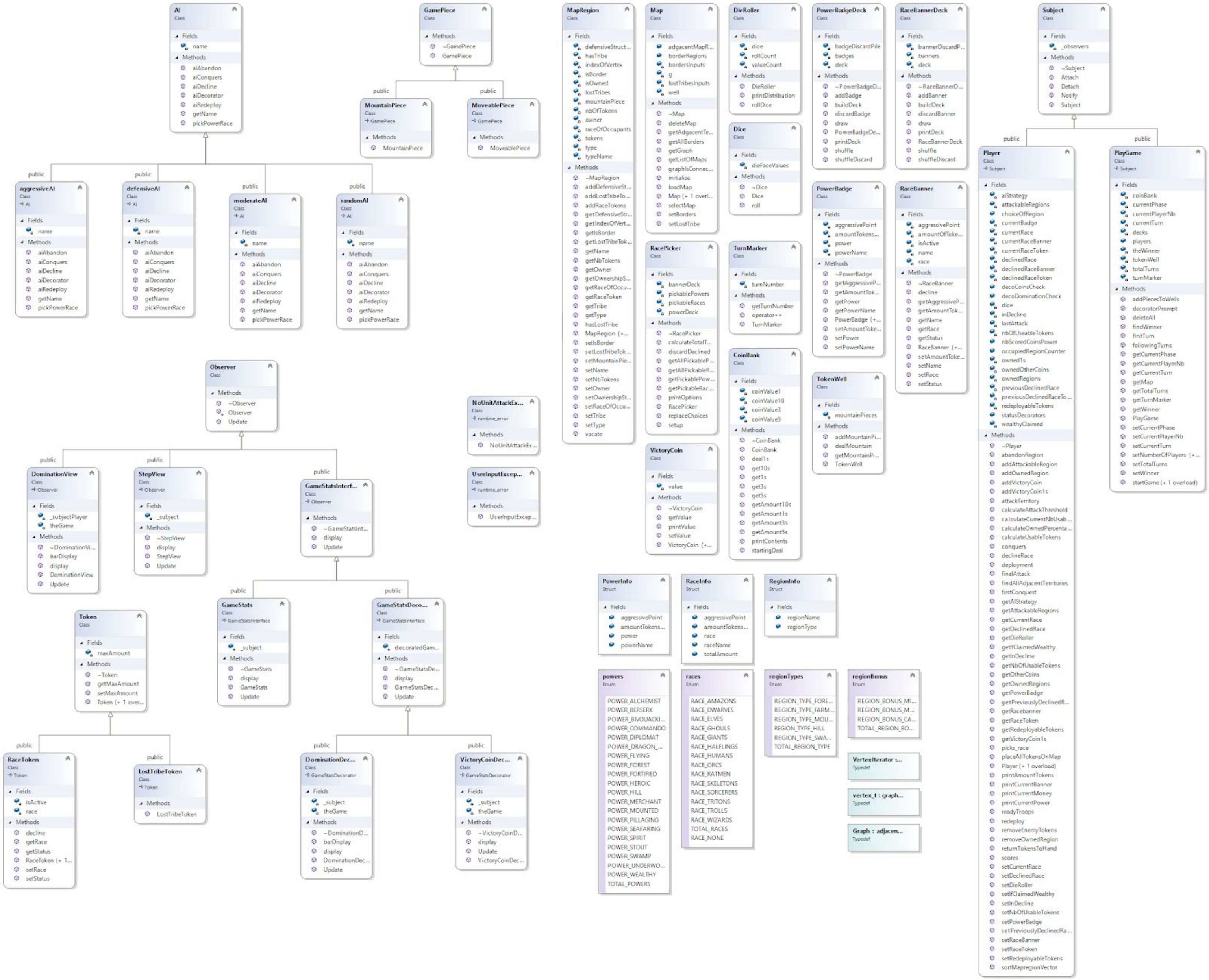
All the objects that need to be created for a game are attributes in the PlayGame class. For example, the raceBanner, powerBadgedeck, a vector of player's, etc.  are attributes in PlayGame class. These attributes are all separately implemented and are used in the PlayGame class.

All player objects hold their victory point score and owned regions as attributes. The player class mainly interacts with the global map object.

For low coupling, we try to have helper functions instead of giant functions. Therefore these smaller functions may be reused in many spots. For example, in Player.cpp, the deploy method is used in redeploy and ready troops.

Additionally, when we modified one module, others were not really affected, because the interfaces (public functions of each class) are how each class communicated with each other.

One instance of high cohesion, is the fact that our classes only do what they are made to do and do not do the functionality of other classes. For example, our mapRegions just give information about what is contained in each map region. Another example is how the player class has all the functionality for actions a player can take and all the information that other modules may want from a player.


AI
Class
Fields
name
Methods
aiAbandon
aiConquers
aiDecline
aiDecorator
aiRedeploy
getName
pickPowerRace
public
aggressiveAI
Class
→ AI
Fields
name
Methods
aiAbandon
aiConquers
aiDecline
aiDecorator
aiRedeploy
getName
pickPowerRace
public
defensiveAI
Class
→ AI
Fields
name
Methods
aiAbandon
aiConquers
aiDecline
aiDecorator
aiRedeploy
getName
pickPowerRace
public
moderateAI
Class
→ AI
Fields
name
Methods
aiAbandon
aiConquers
aiDecline
aiDecorator
aiRedeploy
getName
pickPowerRace
public
randomAI
Class
→ AI
Fields
name
Methods
aiAbandon
aiConquers
aiDecline
aiDecorator
aiRedeploy
getName
pickPowerRace
GamePiece
Class
Methods
~GamePiece
GamePiece
public
MountainPiece
Class
→ GamePiece
Methods
MountainPiece
public
MoveablePiece
Class
→ GamePiece
Methods
MoveablePiece
MapRegion
Class
Fields
defensiveStruct...
hasTribe
indexOfVertex
isBorder
isOwned
lostTribes
mountainPiece
nbOfTokens
owner
raceOfOccupants
tokens
type
typeName
Methods
~MapRegion
addDefensiveSt...
addLostTribeTo...
addRaceTokens
getDefensiveStr...
getIndexOfVert...
getIsBorder
getLostTribeTok...
getName
getNbTokens
getOwner
getOwnershipS...
getRaceOfOccu...
getRaceToken
getTribe
getType
hasLostTribe
MapRegion (+...
setIsBorder
setLostTribeTok...
setMountainPie...
setName
setNbTokens
setOwner
setOwnershipSt...
setRaceOfOccu...
setTribe
setType
vacate
Map
Class
Fields
adgacentMapR...
borderRegions
bordersInputs
g
lostTribesInputs
well
Methods
~Map
deleteMap
getAdgacentTe...
getAllBorders
getGraph
getListOfMaps
graphIsConnec...
initialize
loadMap
Map (+ 1 overl...
selectMap
setBorders
setLostTribe
RacePicker
Class
Fields
bannerDeck
pickablePowers
pickableRaces
powerDeck
Methods
~RacePicker
calculateTotalT...
discardDeclined
getAllPickableP...
getAllPickableR...
getPickablePow...
getPickableRac...
printOptions
RacePicker
replaceChoices
setup
VictoryCoin
Class
Fields
value
Methods
~VictoryCoin
getValue
printValue
setValue
VictoryCoin (+...
DieRoller
Class
Fields
dice
rollCount
valueCount
Methods
DieRoller
printDistribution
rollDice
Dice
Class
Fields
dieFaceValues
Methods
~Dice
Dice
roll
TurnMarker
Class
Fields
turnNumber
Methods
getTurnNumber
operator++
TurnMarker
CoinBank
Class
Fields
coinValue1
coinValue10
coinValue3
coinValue5
Methods
~CoinBank
CoinBank
deal1s
get10s
get1s
get3s
get5s
getAmount10s
getAmount1s
getAmount3s
getAmount5s
printContents
startingDeal
PowerBadgeDeck
Class
Fields
badgeDiscardPile
badges
deck
Methods
~PowerBadgeD...
addBadge
buildDeck
discardBadge
draw
PowerBadgeDe...
printDeck
shuffle
shuffleDiscard
PowerBadge
Class
Fields
aggressivePoint
amountTokens...
power
powerName
Methods
~PowerBadge
getAggressiveP...
getAmountTok...
getPower
getPowerName
PowerBadge (+...
setAmountToke...
setPower
setPowerName
TokenWell
Class
Fields
mountainPieces
Methods
addMountainPi...
dealMountain
getMountainPi...
TokenWell
RaceBannerDeck
Class
Fields
bannerDiscardP...
banners
deck
Methods
~RaceBannerD...
addBanner
buildDeck
discardBanner
draw
printDeck
RaceBannerDeck
shuffle
shuffleDiscard
RaceBanner
Class
Fields
aggressivePoint
amountOfToke...
isActive
name
race
Methods
~RaceBanner
decline
getAggressiveP...
getAmountTok...
getName
getRace
getStatus
RaceBanner (+...
setAmountToke...
setName
setRace
setStatus
Subject
Class
Fields
_observers
Methods
~Subject
Attach
Detach
Notify
Subject
public
Player
Class
→ Subject
Fields
aiStrategy
attackableRegions
choiceOfRegion
currentBadge
currentRace
currentRaceBanner
currentRaceToken
declinedRace
declinedRaceBanner
declinedRaceToken
decoCoinsCheck
decoDominationCheck
dice
inDecline
lastAttack
nbOfUseableTokens
nbScoredCoinsPower
occupiedRegionCounter
owned1s
ownedOtherCoins
ownedRegions
previousDeclinedRace
previousDeclinedRaceTo...
redeployableTokens
statusDecorators
wealthyClaimed
Methods
~Player
abandonRegion
addAttackableRegion
addOwnedRegion
addVictoryCoin
addVictoryCoin1s
attackTerritory
calculateAttackThreshold
calculateCurrentNbUsab...
calculateOwnedPercenta...
calculateUsableTokens
conquers
declineRace
deployment
finalAttack
findAllAdjacentTerritories
firstConquest
getAIStrategy
getAttackableRegions
getCurrentRace
getDeclinedRace
getDieRoller
getIfClaimedWealthy
getInDecline
getNbOfUsableTokens
getOtherCoins
getOwnedRegions
getPowerBadge
getPreviouslyDeclinedR...
getRacebanner
getRaceToken
getRedeployableTokens
getVictoryCoin1s
picks_race
placeAllTokensOnMap
Player (+ 1 overload)
printAmountTokens
printCurrentBanner
printCurrentMoney
printCurrentPower
readyTroops
redeploy
removeEnemyTokens
removeOwnedRegion
returnTokensToHand
scores
setCurrentRace
setDeclinedRace
setDieRoller
setIfClaimedWealthy
setInDecline
setNbOfUsableTokens
setPowerBadge
setPreviouslyDeclinedRa...
setRaceBanner
setRaceToken
setRedeployableTokens
sortMapregionVector
public
PlayGame
Class
→ Subject
Fields
coinBank
currentPhase
currentPlayerNb
currentTurn
decks
players
theWinner
tokenWell
totalTurns
turnMarker
Methods
addPiecesToWells
decoratorPrompt
deleteAll
findWinner
firstTurn
followingTurns
getCurrentPhase
getCurrentPlayerNb
getCurrentTurn
getMap
getTotalTurns
getTurnMarker
getWinner
PlayGame
setCurrentPhase
setCurrentPlayerNb
setCurrentTurn
setNumberOfPlayers (+...
setTotalTurns
setWinner
startGame (+ 1 overload)
Observer
Class
Methods
~Observer
Observer
Update
public
DominationView
Class
→ Observer
Fields
_subjectPlayer
theGame
Methods
~DominationVi...
barDisplay
display
DominationView
Update
public
StepView
Class
→ Observer
Fields
_subject
Methods
~StepView
display
StepView
Update
public
GameStatsInterf...
Class
→ Observer
Methods
~GameStatsInt...
display
Update
public
GameStats
Class
→ GameStatsInterface
Fields
_subject
Methods
~GameStats
display
GameStats
Update
public
GameStatsDeco...
Class
→ GameStatsInterface
Fields
decoratedGam...
Methods
~GameStatsDe...
display
GameStatsDec...
Update
public
DominationDec...
Class
→ GameStatsDecorator
Fields
_subject
theGame
Methods
~DominationD...
barDisplay
display
DominationDec...
Update
public
VictoryCoinDec...
Class
→ GameStatsDecorator
Fields
_subject
theGame
Methods
~VictoryCoinD...
display
Update
VictoryCoinDec...
Token
Class
Fields
maxAmount
Methods
~Token
getMaxAmount
setMaxAmount
Token (+ 1 over...
public
RaceToken
Class
→ Token
Fields
isActive
race
Methods
decline
getRace
getStatus
RaceToken (+ 1...
setRace
setStatus
public
LostTribeToken
Class
→ Token
Methods
LostTribeToken
NoUnitAttackEx...
Class
→ runtime_error
Methods
NoUnitAttackEx...
UserInputExcep...
Class
→ runtime_error
Methods
UserInputExcep...
PowerInfo
Struct
Fields
aggressivePoint
amountTokens...
power
powerName
RaceInfo
Struct
Fields
aggressivePoint
amountTokens...
race
raceName
totalAmount
RegionInfo
Struct
Fields
regionName
regionType
powers
Enum
POWER_ALCHEMIST
POWER_BERSERK
POWER_BIVOUACKI...
POWER_COMMANDO
POWER_DIPLOMAT
POWER_DRAGON_...
POWER_FLYING
POWER_FOREST
POWER_FORTIFIED
POWER_HEROIC
POWER_HILL
POWER_MERCHANT
POWER_MOUNTED
POWER_PILLAGING
POWER_SEAFARING
POWER_SPIRIT
POWER_STOUT
POWER_SWAMP
POWER_UNDERWO...
POWER_WEALTHY
TOTAL_POWERS
races
Enum
RACE_AMAZONS
RACE_DWARVES
RACE_ELVES
RACE_GHOULS
RACE_GIANTS
RACE_HALFLINGS
RACE_HUMANS
RACE_ORCS
RACE_RATMEN
RACE_SKELETONS
RACE_SORCERERS
RACE_TRITONS
RACE_TROLLS
RACE_WIZARDS
TOTAL_RACES
RACE_NONE
regionTypes
Enum
REGION_TYPE_FORE...
REGION_TYPE_FARM...
REGION_TYPE_MOU...
REGION_TYPE_HILL
REGION_TYPE_SWA...
TOTAL_REGION_TYPE
regionBonus
Enum
REGION_BONUS_MI...
REGION_BONUS_M...
REGION_BONUS_CA...
TOTAL_REGION_BO...
VertexIterator :...
Typedef
vertex_t : graph...
Typedef
Graph : adjacen...
Typedef

## **Part 4**

| Concept used | Brief definition/description | Give: class/function/file name including extension |
|---|---|---|
| Pointer/smart pointers | A pointer is an object, whose value refers to another values memory address / Smart pointers function the same as normal pointers, but they have reduce bugs, retain efficiency and control memory management by the use of automated methods | SmallWorldApp/PlayGame/PlayGame.cpp<br>void PlayGame::firstTurn() |
| Memory management | Memory management is how the memory can be managed with garbage collection. It can be managed automatically by the use of smart pointers or manually with the use of the "new" and "delete" reserved words in c++. | SmallWorldApp/PlayGame/PlayGame.cpp<br>PlayGame class<br>void PlayGame::deleteAll() |
| Vectors | Similar to an array but it's size can increase and decrease as needed. Using "at" and "size" prevent subscript range errors. | SmallWorldApp/Headers/AI.h<br>From aggressiveAi class<br>int aiConquers(Player* aiPlayer, vector<MapRegion*> regions) |
| Data structure | A data structure is a system that can efficiently store and modify data in memory. | SmallWorldApp/Headers/Tokens.h<br>PowerBadgeDeck class<br>std::queue<PowerBadge*> deck; |

| | | |
|---|---|---|
| Operator overloading | It is polymorphism where an operator have a different implementation depending on its arguments. Example: we can overload the implementation of “<” to have a way to sort objects of a certain class. | SmallWorldApp/Headers/TurnMarker.h<br>In the TurnMarker’s header<br>void operator++() { ++turnNumber; } |
| File I/O | It is a way to input some data in a file or to output data from a file. | SmallWorldApp/Map/Map.cpp<br>void Map::loadMap(string filename) |
| Exception handling | It is a mechanism that allows to communicate to the user when a program has encountered an anomaly during the execution of a program. Also, to decide what the program will do after the error occurred. | SmallWorldApp/Players/Player.cpp<br>Player class<br>int Player::attackTerritory(Map Region *region) |
| Templates | Templates in c++ is a feature that allows functions to use generic types. This allows the function to be used with any type of data. | Anytime we use lists or vectors, we are using templates<br>SmallWorldApp/Headers/Subject.h<br>std::list<Observer*> *_observers; |
| Any library including GUI | It’s many files that simplify coding by giving useful tools such as functions and classes. It includes development tools to build a Graphical User Interface. | SmallWorldApp/Map/Map.cpp<br>bool Map::graphIsConnected()<br>We used boost library |