

Diagram Assembly Document

Drew, Keith
`keithd@vandals.uidaho.edu`

Jaskowiak, Tyler
`jasz7989@vandals.uidaho.edu`

Pearhill, Gabe
`pear9915@vandals.uidaho.edu`

Klingenberg, David
`bigwookie@gmail.com`

Fuhrman, Wayne
`fuhr0438@vandals.uidaho.edu`

Goes, Chris
`goes8945@vandals.uidaho.edu`

February 18, 2014

1 Dictionary

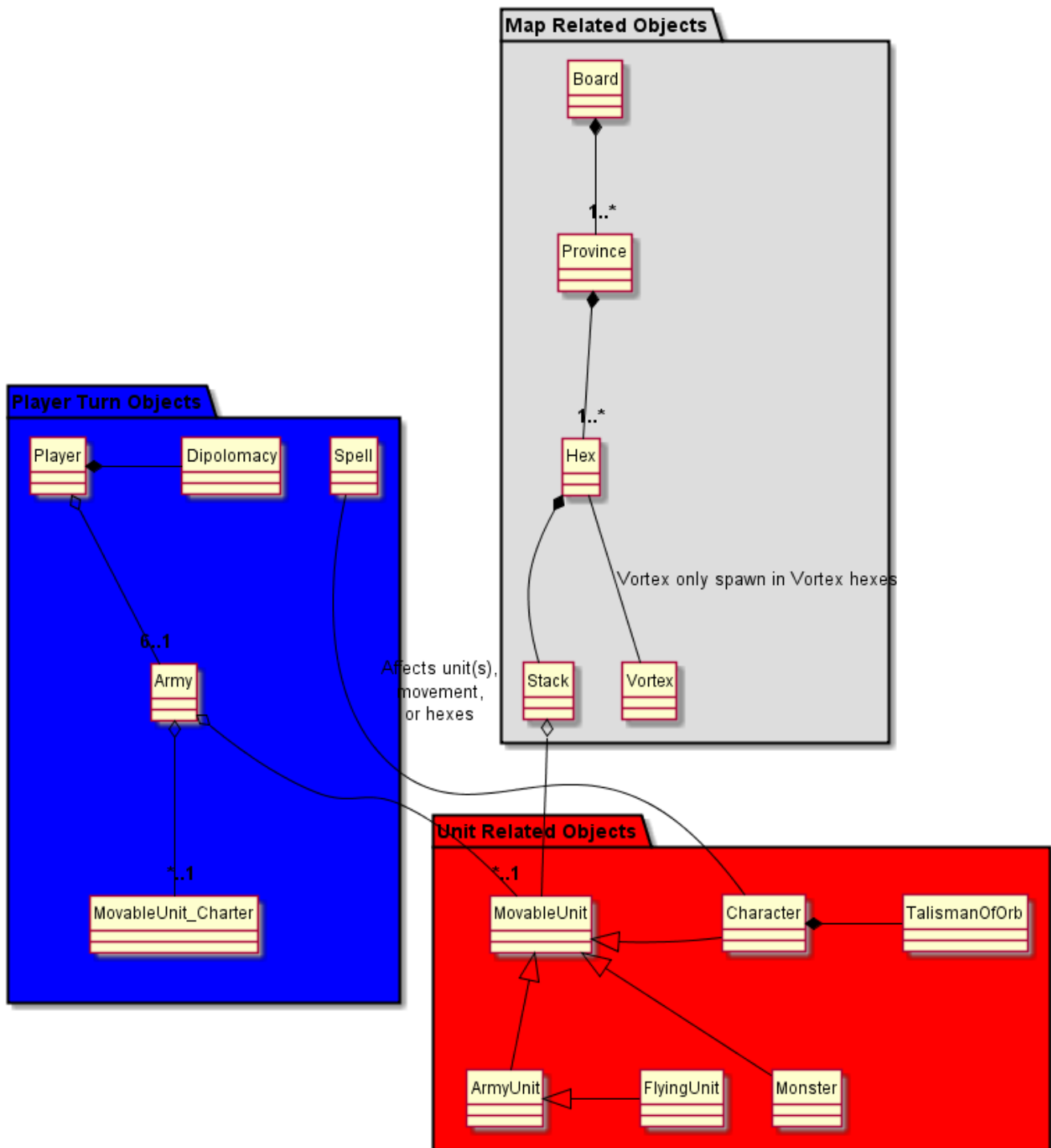
- Board: The game board, this holds nearly all visual information about the game and state. The class holds random event flags, which will become more specific, that indicate global events.
- Province: A province is a group of hexes that are related by controlling faction. Essentially like the United States of Hexes.
- Hex: A discrete location on the board. Hexes are represented by a unique ID, terrain, and a stack of units that can be empty.
- Stack: A collection of units and characters, bound by the rules of the game. Ie, 0 or more characters, and 0-2 units. Also, special considerations for movement phase, flying units, and monsters.
- Special Hex: The special hex can be any of the residential hexes (city/capital/town/castle), or it can be one of the several unique hex types, such as a teleport hex or vortex hex, or the terrifying Bottomless Plungehole.
- BottomlessPlungehole: The bottomless plungehole is a special hex that destroys all non-flying units passing over it or any characters & flying units who end their movement on this hex. Therefore, it has a method for discriminantly destroying a stack.
- ResidentialHex: Since cities, capitals, towns, and castles have inherent defense rating and leadership, they have fields and methods relating to these properties. As defenseRating can change, the initial value is also stored so that a user can regenerate it.
- TeleportHex The portal hexes, in addition to other properties of hexes, can also teleport units and characters to other portals.
- VortexHex: These impassable hexes are the spawn points of the treacherous vortices that run rampant throughout the planet damaging units.
- Vortex: A vortex is a moveable unit, from the system's point of view. A character can control vortices under certain conditions. Otherwise movement, creation, and destruction of vortices is automated.
- Diplomacy: An object defining the relationship between players and neutral entities.

- Player: The human player. This object contains information about the players armies, diplomatic relations, race, and victory points.
- Army: This object is responsible for conveniently managing the units of each individual diplomatic entity.
- Spells: An object that contains the stats and effects of each castable spell.
- CounterSpells: An extension of the spell class, with the added properties of counterspells.
- VictoryConditions: A class to manage and check for victory conditions.
- Scenario: A class to manage the specific scenario conditions and story elements. Will initialize victory conditions.
- Alliances: A class to facilitate player interactions and diplomatic relations.
- PreTurnPhase: A class to initialize the game turn.
- PlayerTurnPhase: A class to initialize the player turn phase.
- PostTurnPhase: A class to handle post-turn house keeping.
- RandomEvent: This object is responsible for facilitating random events.
- RandomMovement: This object is responsible for handling the movement of neutral characters on the game board.
- PlayerOrderDetermination: Responsible for finding the next player.
- GameSetUp: A class the select the scenario and handle initial unit placement.
- VictoryConditions: A class to manage and check for victory conditions.
- Scenario: A class to manage the specific scenario conditions and story elements. Will initialize victory conditions.
- SwordSorcery: A class to handle the main menu, loading, and saving games.
- MannaRegeneration: An object to handle the regeneration of manna based on the stellar configuration for the Characters on the game board.
- MovableUnit: Something that can moved by the player.

2 Team A Class Diagram

Class Over View of Sword & Sorcery

Author: David Klingenberg

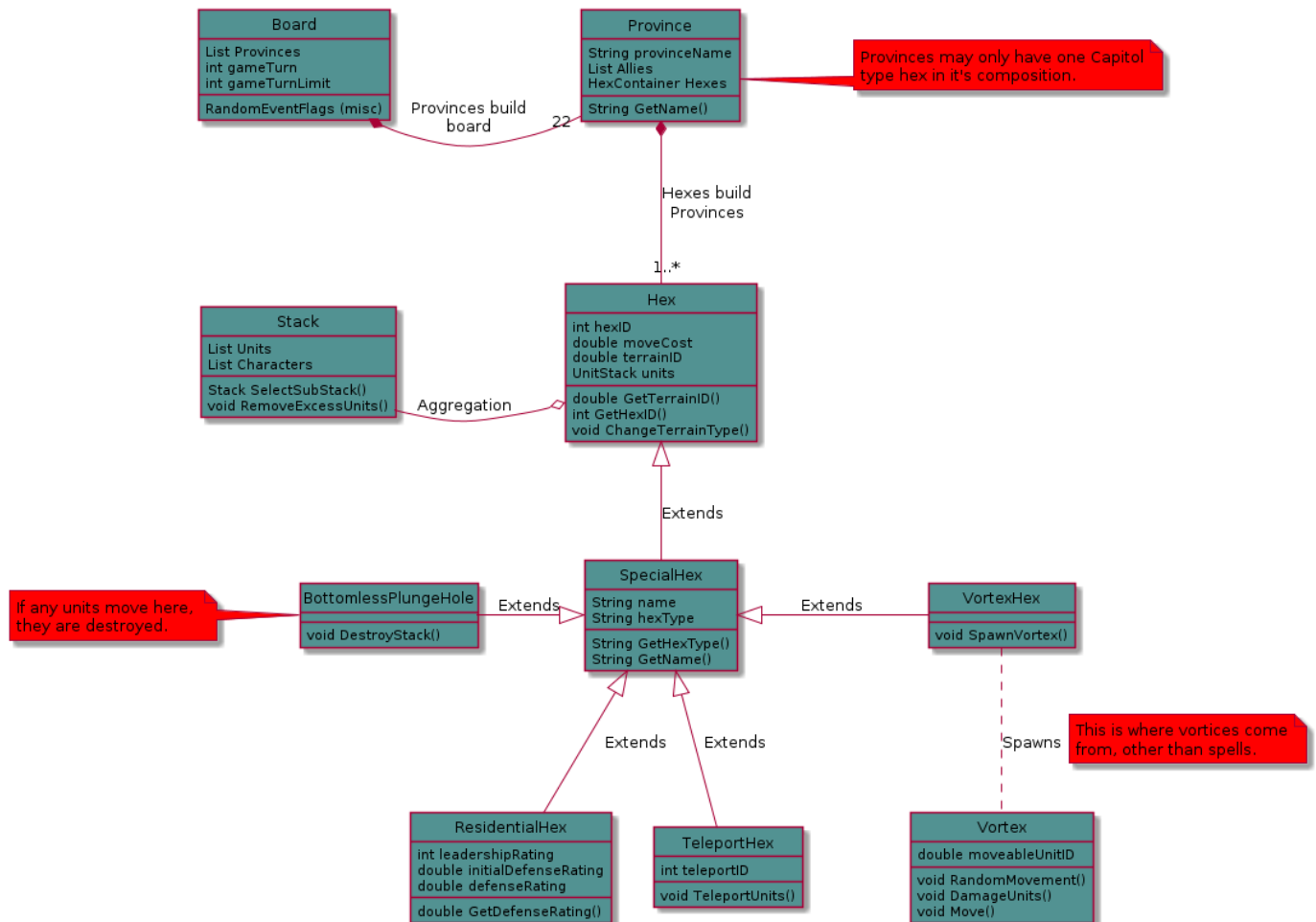


2.1 Team A PlantUML Source

3 Subteam Diagrams

3.1 Keith and Tyler - Reviewed by Wayne

Keith Drew & Tyler Jaskowskiak Class Diagrams
Map Related



3.1.1 PlantUML Source

```
@startuml
title Keith Drew \& Tyler Jaskowskiak Class Diagrams\nMap Related
hide circle
class Board #529292 {
    List Provinces
    RandomEventFlags (misc)
    int gameTurn
    int gameTurnLimit
}
class Province #529292 {
    String provinceName
    String GetName()
    List Allies
    HexContainer Hexes
}
```

```

note right of Province #red
    Provinces may only have one Capitol
    type hex in it's composition.
end note
class Hex #529292 {
    int hexID
    double moveCost
    double terrainID
    UnitStack units
    double GetTerrainID()
    int GetHexID()
    void ChangeTerrainType()
}
class Stack #529292 {
    List Units
    List Characters
    Stack SelectSubStack()
    void RemoveExcessUnits()
}
class SpecialHex #529292 {
    String name
    String hexType
    String GetHexType()
    String GetName()
}
class ResidentialHex #529292 {
    int leadershipRating
    double initialDefenseRating
    double defenseRating
    double GetDefenseRating()
}
class BottomlessPlungeHole #529292 {
    void DestroyStack()
}
note left of BottomlessPlungeHole #red
    If any units move here,
    they are destroyed.
end note
class VortexHex #529292 {
    void SpawnVortex()
}
class TeleportHex #529292 {
    void TeleportUnits()
    int teleportID
}
class Vortex #529292 {
    double moveableUnitID
    void RandomMovement()
    void DamageUnits()
    void Move()
}
Board *-right- "22" Province : Provinces build\nboard
Province *-down- "1..*" Hex : Hexes build\nProvinces

```

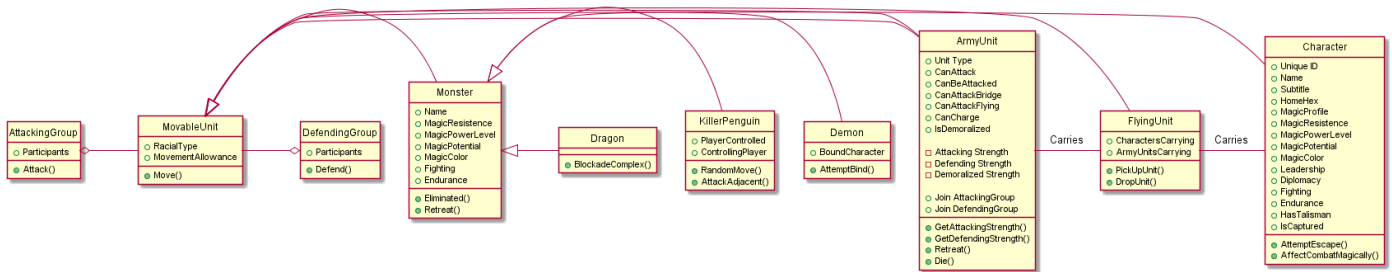
```

Hex <|-down- SpecialHex : Extends
Vortex .up. VortexHex : Spawns
note right on link #red
    This is where vortices come
    from, other than spells.
end note
TeleportHex --up|> SpecialHex : Extends
VortexHex --left|> SpecialHex : Extends
BottomlessPlungeHole --right|> SpecialHex : Extends
ResidentialHex --up|> SpecialHex : Extends
Stack -leftto Hex : Aggregation
@enduml

```

4 Detail Diagrams

4.1 Wayne And Chris- Reviewed by Gabe and David



- A movable unit is something with a race that a player can move.
- An army unit is something that attacks and defends directly. Army units are mostly differentiated by their fixed attacking and defending strengths, and by restrictions on what sort of units and terrain they can attack. Each individual army unit may be in several different states. IsDemoralized is a state that can persist between game turns. CanAttack and CanBeAttacked are states that are reset each game turn, (these states stem from the rule that each unit can only participate in a single attack each game turn).
- What can an ArmyUnit do? Attack, defend, retreat and die. The games rules are best reflected by considering attacking and defending as group operations (possibly in a group of size 1). The AttackingGroup and DefendingGroup classes capture this.
- FlyingUnits are a subclass of normal ArmyUnits, but they can carry a single unit or any number of characters.

4.1.1 PlantUML Source

```

@startuml
hide circle

MovableUnit <|-- ArmyUnit
MovableUnit <|-- Character
MovableUnit <|-- Monster
FlyingUnit -|> MovableUnit
Monster <|-- Demon
Monster <|-- KillerPenguin
Monster <|-- Dragon

```

FlyingUnit -- Character : Carries

ArmyUnit -- FlyingUnit : Carries

AttackingGroup o- MovableUnit

MovableUnit -o DefendingGroup

```
class AttackingGroup{
+ Participants
+ Attack()
}
```

```
class DefendingGroup{
+ Participants
+ Defend()
}
```

```
class ArmyUnit {

+ Unit Type
+ CanAttack
+ CanBeAttacked
+ CanAttackBridge
+ CanAttackFlying
+ CanCharge
+ IsDemoralized

- Attacking Strength
- Defending Strength
- Demoralized Strength

+ GetAttackingStrength()
+ GetDefendingStrength()
+ Join AttackingGroup
+ Join DefendingGroup
+ Retreat()
+ Die()

}
```

```
class FlyingUnit {
+ CharactersCarrying
+ ArmyUnitsCarrying
+ PickUpUnit()
+ DropUnit()
}
```

```
class MovableUnit {
+ RacialType
+ MovementAllowance
+ Move()
}
```

```

class Character {
+ Unique ID
+ Name
+ Subtitle
+ HomeHex
+ MagicProfile
+ MagicResistance
+ MagicPowerLevel
+ MagicPotential
+ MagicColor
+ Leadership
+ Diplomacy
+ Fighting
+ Endurance
+ HasTalisman
+ IsCaptured

+ AttemptEscape()
+ AffectCombatMagically()
}

```

```

class Monster {
+ Name
+ MagicResistance
+ MagicPowerLevel
+ MagicPotential
+ MagicColor
+ Fighting
+ Endurance

+ Eliminated()
+ Retreat()
}

```

```

class Demon {
+ BoundCharacter

+ AttemptBind()
}

```

```

class KillerPenguin {
+ PlayerControlled
+ ControllingPlayer

+ RandomMove()
+ AttackAdjacent()

}

```

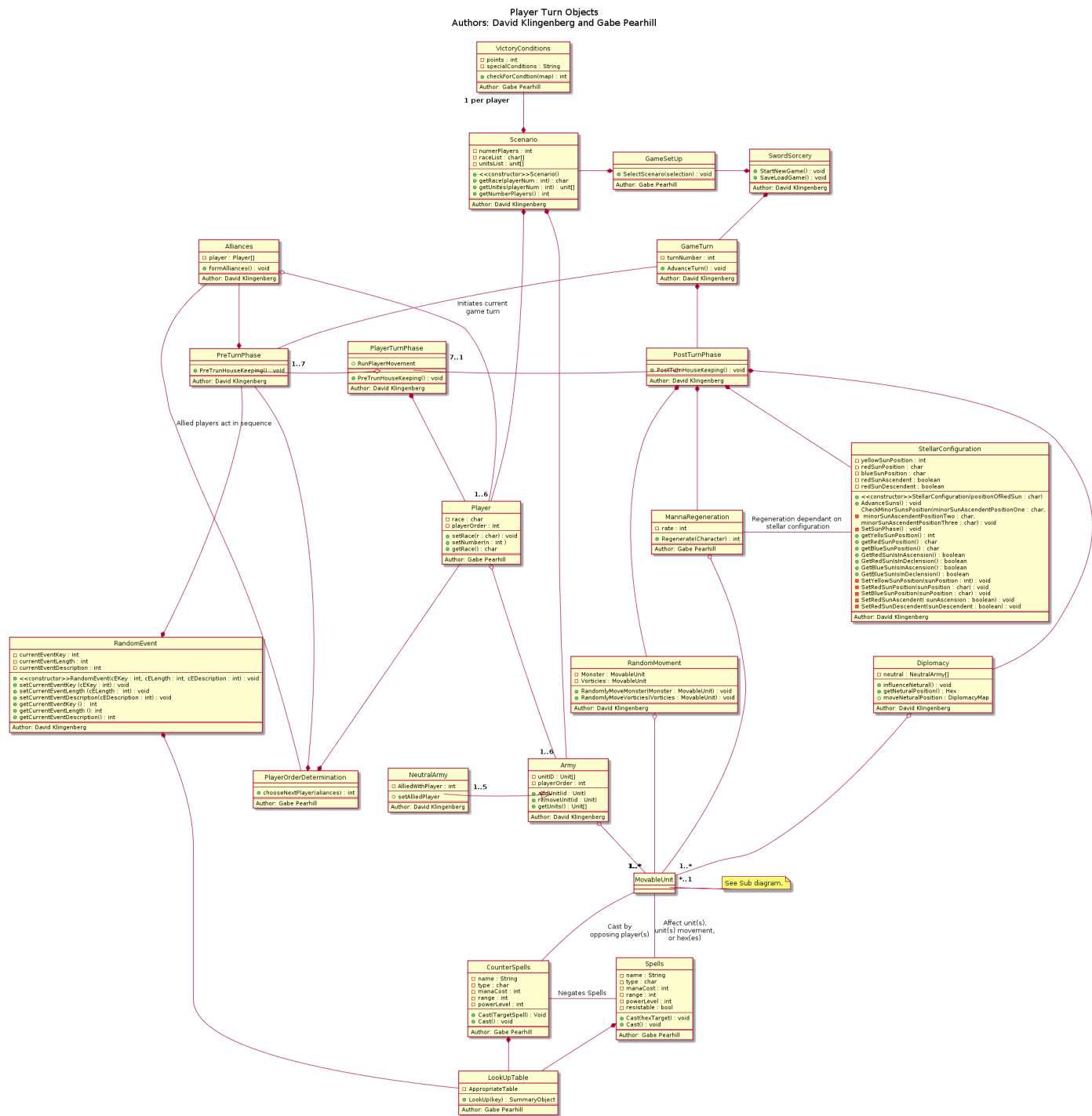
```

class Dragon {
+ BlockadeComplex()
}

@enduml

```


4.2 David and Gabe - Reviewed by Keith and Tyler



4.2.1 PlantUML Source

```
@startuml
hide circles
```

```
title Player Turn Objects\nAuthors: David Klingenberg and Gabe Pearhill
```

```
class Player{
-race : char
-playerOrder : int
--
+setRace(r : char) : void
+setNumber(n : int )
+getRace() : char
```

```

==
Author: Gabe Pearhill
}

class Army{
-unitID : Unit[]
-playerOrder : int
--
+addUnit(id : Unit)
+removeUnit(id : Unit)
+getUnits() : Unit[]
==
Author: David Klingenberg
}

class NeutralArmy{
-AlliedWithPlayer : int
--
+setAlliedPlayer
==
Author: David Klingenberg
}

class StellarConfiguration{
-yellowSunPosition : int
-redSunPosition : char
-blueSunPosition : char
-redSunAscendent : boolean
-redSunDescendent : boolean
--
+<<constructor>>StellarConfiguration(positionOfRedSun : char)
+AdvanceSuns() : void
-CheckMinorSunsPosition(minorSunAscendentPositionOne : char,\n minorSunAscendentPositionTwo
-SetSunPhase() : void
+getYelloSunPosition() : int
+getRedSunPosition() : char
+getBlueSunPosition() : char
+GetRedSunIsInAscension() : boolean
+GetRedSunIsInDeclension() : boolean
+GetBlueSunIsInAscension() : boolean
+GetBlueSunIsInDeclension() : boolean
-SetYellowSunPosition(sunPosition : int) : void
-SetRedSunPosition(sunPosition : char) : void
-SetBlueSunPosition(sunPosition : char) : void
-SetRedSunAscendent( sunAscension : boolean) : void
-SetRedSunDescendent(sunDescendent : boolean) : void
==
Author: David Klingenberg
}

class Scenario {
-numerPlayers : int
-raceList : char[]

```

```

-unitsList : unit[]
--
+<<constructor>>Scenario()
+getRace(playerNum : int) : char
+getUnites(playerNum : int) : unit[]
+getNumberPlayers() : int
==
Author: David Klingenberg
}

class RandomEvent{
-currentEventKey : int
-currentEventLength : int
-currentEventDescription : int
--
+<<constructor>>RandomEvent(cEKey : int, cELength : int, cEDescription : int) : void
+setCurrentEventKey (cEKey : int) : void
+setCurrentEventLength (cELength : int) : void
+setCurrentEventDescription(cEDescription : int) : void
+getCurrentEventKey () : int
+getCurrentEventLength (): int
+getCurrentEventDescription() : int
==
Author: David Klingenberg
}

class Diplomacy{
-neutral : NeutralArmy[]
--
+influenceNetural() : void
+getNeturalPosition() : Hex
+moveNeturalPosition : DiplomacyMap
==
Author: David Klingenberg
}

class PreTurnPhase{
--
+PreTrunHouseKeeping() : void
==
  Author: David Klingenberg
}

class PlayerTurnPhase{
--
+RunPlayerMovement
==
--
+PreTrunHouseKeeping() : void
==

```

```

    Author: David Klingenberg
}

class PostTurnPhase{
--
+PostTurnHouseKeeping() : void
==
    Author: David Klingenberg
}

class GameTurn{
-turnNumber : int
--
+AdvanceTurn() : void
==
    Author: David Klingenberg
}

class Spells{
-name : String
-type : char
    -manaCost : int
    -range : int
    -powerLevel : int
    -resistable : bool
    --
    +Cast(hexTarget) : void
    +Cast() : void
    ==
    Author: Gabe Pearhill
}

class CounterSpells{
-name : String
-type : char
    -manaCost : int
    -range : int
    -powerLevel : int
    --
+Cast(TargetSpell) : Void
+Cast() : void
==
    Author: Gabe Pearhill
}

class Alliances{
-player : Player[]
--
+formAlliances() : void
==
    Author: David Klingenberg
}

class SwordSorcery{

```

```

--
+StartNewGame() : void
+SaveLoadGame() : void
==
Author: David Klingenberg
}

class PlayerOrderDetermination{
--
+chooseNextPlayer(aliases) : int
==
Author: Gabe Pearhill
}
class Alliances
class SwordSorcery
class MannaRegeneration {
-rate : int
--
+Regenerate(Character) : int
==
Author: Gabe Pearhill
}

class MovableUnit{
}
note right : See Sub diagram.

class RandomMovment{
-Monster : MovableUnit
-Vorticies : MovableUnit
--
+RandomlyMoveMonster(Monster : MovableUnit) : void
+RandomlyMoveVorticies(Vorticies : MovableUnit) : void
==
Author: David Klingenberg
}

Class GameSetUp{
--
+SelectScenaro(selection) : void
==
Author: Gabe Pearhill
}

class VictoryConditions{
-points : int
-specialConditions : String
--
+checkForCondtion(map) : int
==
Author: Gabe Pearhill
}

```

```

class LookUpTable{
-AppropriateTable
--
+LookUp(key) : SummaryObject
==
Author: Gabe Pearhill
}

SwordSorcery *-- GameTurn
GameSetUp -* SwordSorcery
Scenario -* GameSetUp
VictoryConditions "<b>1 per player</b>" --* Scenario

Scenario *-- Player
Scenario *-- Army

GameTurn *-- PostTurnPhase
GameTurn -- PreTurnPhase : Initiates current\ngame turn

PlayerTurnPhase <b>7..1</b> -o PostTurnPhase
PreTurnPhase <b>1..7</b> -o PlayerTurnPhase

PreTurnPhase --* RandomEvent
PlayerOrderDetermination *-- PreTurnPhase
Alliances --* PreTurnPhase
Alliances -- PlayerOrderDetermination : Allied players act in sequence

PlayerTurnPhase *-- Player
Player o-- <b>1..6</b> Army
Army o-- <b>1..*</b> MovableUnit
NeutralArmy <b>1..5</b> -|> Army

MovableUnit -- Spells : Affect unit(s),\nunit(s) movement,\nor hex(es)
MovableUnit -- CounterSpells : Cast by\nopposing player(s)
CounterSpells - Spells : Negates Spells

Spells *-- LookUpTable
CounterSpells *-- LookUpTable
RandomEvent *-- LookUpTable

PostTurnPhase *-- RandomMovment
StellarConfiguration --* PostTurnPhase
StellarConfiguration MannaRegeneration : Regeneration dependant on\n stellar configuration
PostTurnPhase *-- Diplomacy
PostTurnPhase *-- MannaRegeneration

Alliances o-- <b>1..6</b> Player
Player --* PlayerOrderDetermination

RandomMovment o-- <b>1..*</b> MovableUnit
MannaRegeneration o-- <b>1..*</b> MovableUnit
Diplomacy o-- <b>*.1</b> MovableUnit

```

@endum1