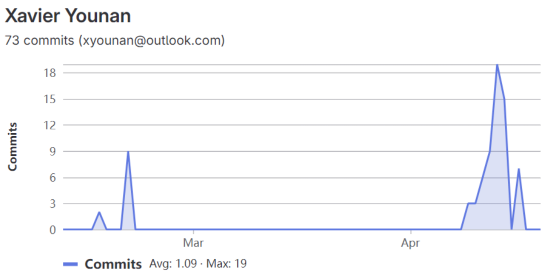
FIT3077
Sprint 3 Documentation

Team XDCA

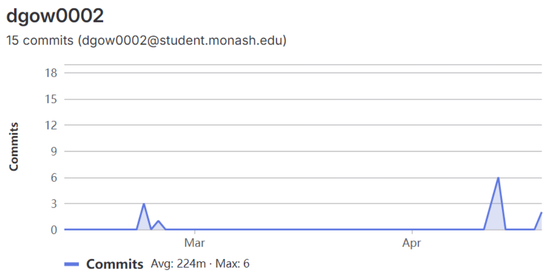
Table of Contents

Contributor Analytics



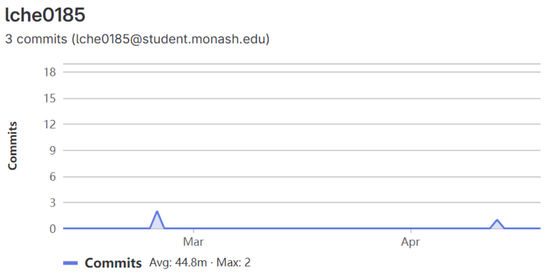
Xavier Younan

73 commits (xyounan@outlook.com)



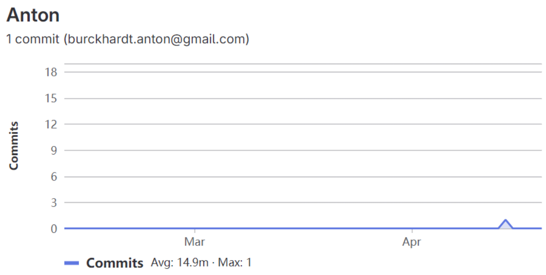
dgow0002

15 commits (dgow0002@student.monash.edu)



lche0185

3 commits (lche0185@student.monash.edu)



Anton

1 commit (burckhardt.anton@gmail.com)

Sprint 2 Review

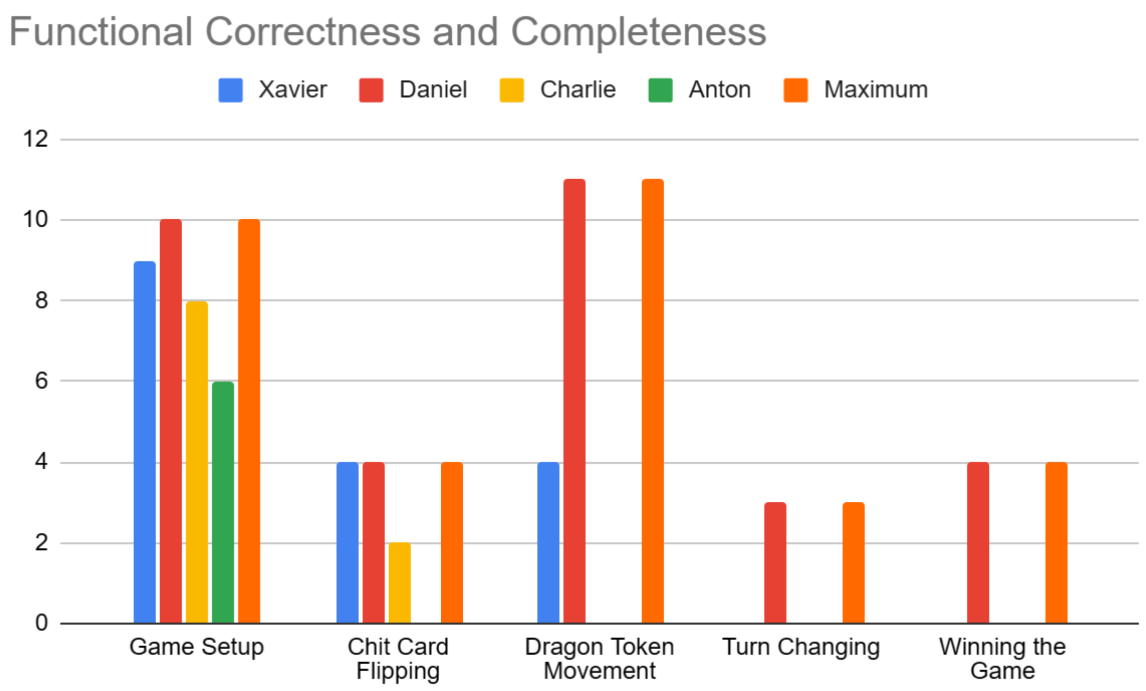
At the start of Sprint 3 the team conducted a meeting with all members in attendance to collaborate on determining valid metrics to evaluate the Sprint 2 designs and codebases based on the 7 provided IEEE quality model sub-characteristics [1] (2 of which have been grouped). The metrics, their justification, and results are provided in the Results & Justification subsections below with each subsection focusing on one of the provided sub-characteristics.

Results

Functional Completeness & Correctness

Functional Completeness and Correctness depends heavily on the specific problem being solved, as a result there are no high-quality standardised metrics in the literature to apply here. Instead, as a team, we broke every functional category from the Sprint 2 documentation down into atomic subcategories such that as much of the intended game features and nuances were covered as possible. These categories included Game Setup, Chit Card Flipping, Dragon Token/Player Movement, Turn Changing, and Winning the Game. Subcategories are provided in Appendix A.

Each subcategory was then given a 1 point allocation and the major categories were graded as a sum of the points achieved. The results of which can be seen in the Figure below.



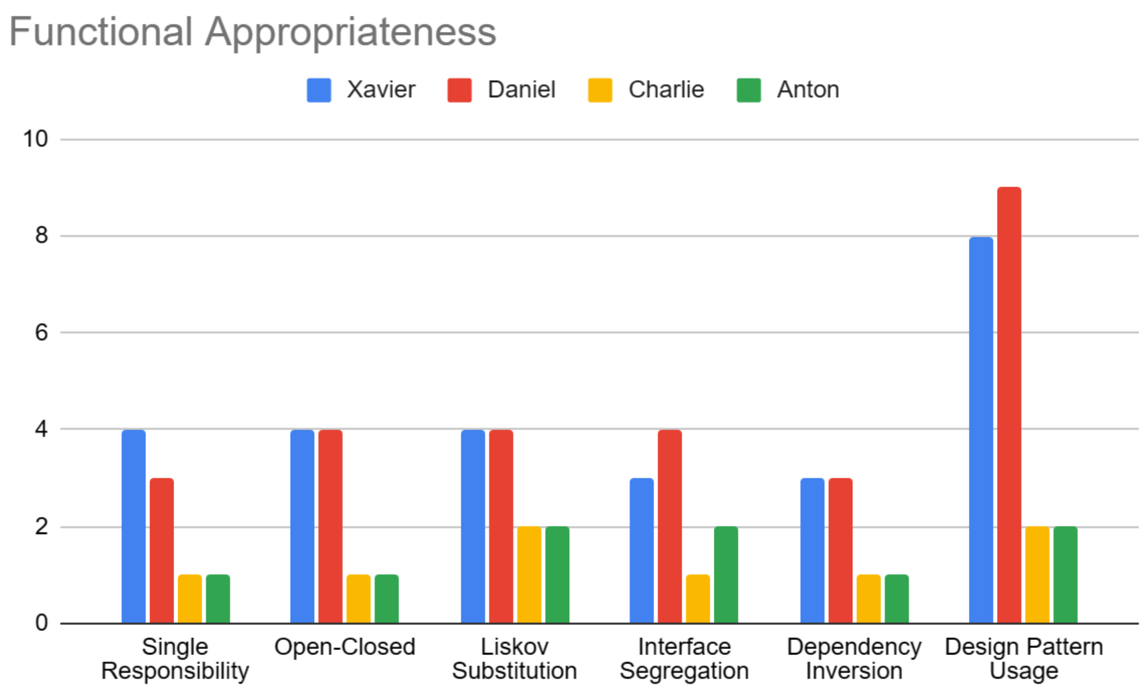
Functional Appropriateness

Functional Appropriateness is concerned with the “degree to which functions facilitate the accomplishment of specified tasks and objectives.” As FIT3077 is a heavily Object-Oriented focused unit, we decided to grade appropriateness based on an assessment of the SOLID principles and an evaluation of the usage of Design Patterns in terms of Appropriateness, Effectiveness and Extensibility.

As there is no clear mathematical expression for any of the SOLID principles or the design pattern evaluation categories, we had to determine a method of grading these ourselves. We decided to allocate 5 points to each of the key categories to be graded. For example, the Single Responsibility Principle in the SOLID principle can be graded out of 5, and Appropriateness of Design Pattern usage is also graded out of 5. This is a consistent decision we adhered to when determining metrics for subjective measures and it will be seen consistently throughout the other proposed metrics.

On their own, subjective measures open up an avenue for bias and misevaluation. However, we concluded that, given our existing IT experience gained throughout multiple university Units, we could likely provide a “close-enough” evaluation here that will guide our future decisions.

The results can be seen in the Figure below.



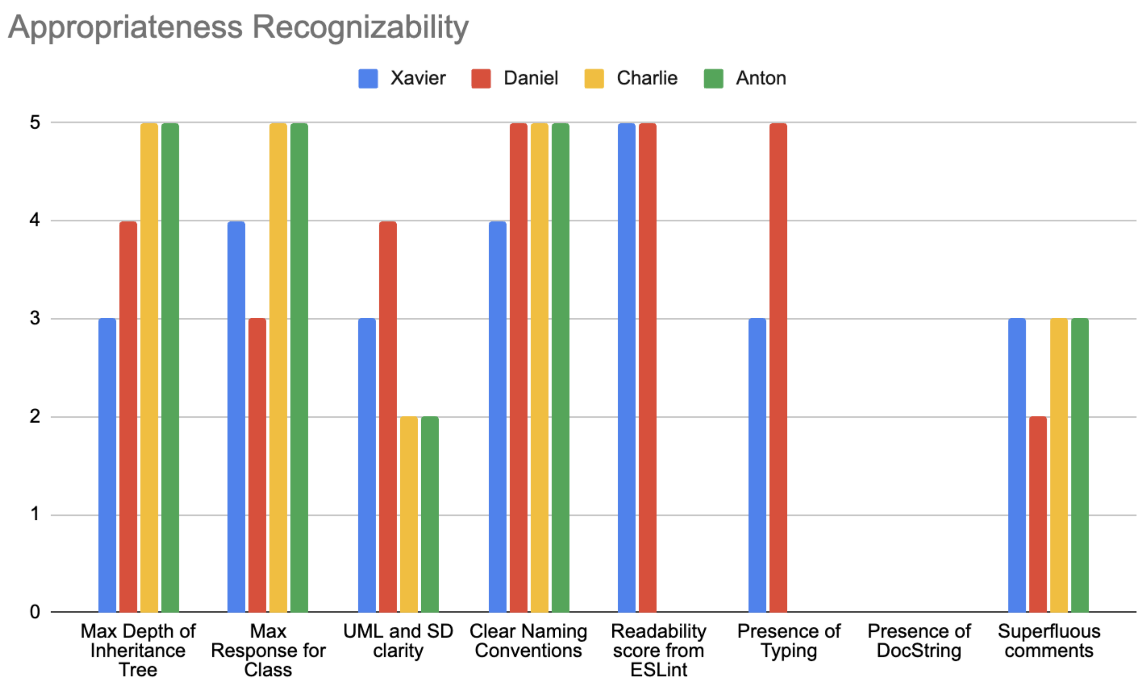
Appropriateness Recognizability

Appropriateness recognizability revolves around how well one can understand whether the built system handles all the key game functionalities (i.e. the solution direction). This can also end up becoming more subjective, as the understanding of aspects varies from individual to individual, however to combat this we decided to structure our grading system into key aspects of the code. Here, we continued with our consistent metric of grading each aspect out of 5 to allow ample room to reflect the subjective grade.

Max depth of inheritance trees and max response for a class (referring to the amount of methods called from another class), as well as UML and SD clarity, revolved around the general complexity entailed within the system. As stated in Chidamber and Kemerer article [2], the deeper the hierarchy level, the more methods are inherited and the more complex it is to predict the systems core solution direction. This is also closely linked to the maximum methods called from another class, as this can also lead to it becoming harder to understand the system. Additionally the UML and SD clarity, although subjective at times, play a clear role in this field too. It is worth noting that the max depth of inheritance and max response for a class had a reverse grading system with it being marked down the higher the complexity to properly reflect the understandings of Chidamber and Kemerer.

The understandability of the code on a more micro level also significantly influences the clarity and how well one can assess the appropriateness of the applied code. Here we agreed as a team that creating markings for clear naming conventions, presence of typing, presence of DocString, superfluous comments and the readability score of ESLint covered all bases to properly encapsulate the evaluation revolving around understanding the core solution direction on a smaller level too.

The results can be found in the Figure below.



Modifiability

For modifiability we used a mix of subjective and literature supported objective measures for evaluation. Specifically, we made use of the highly regarded Chidamber and Kemerer journal article “A metrics suite for object oriented design” [2]. According to their evaluation of the proposed metrics we chose to use “Maximum Number of Children” as the primary metric from this paper.

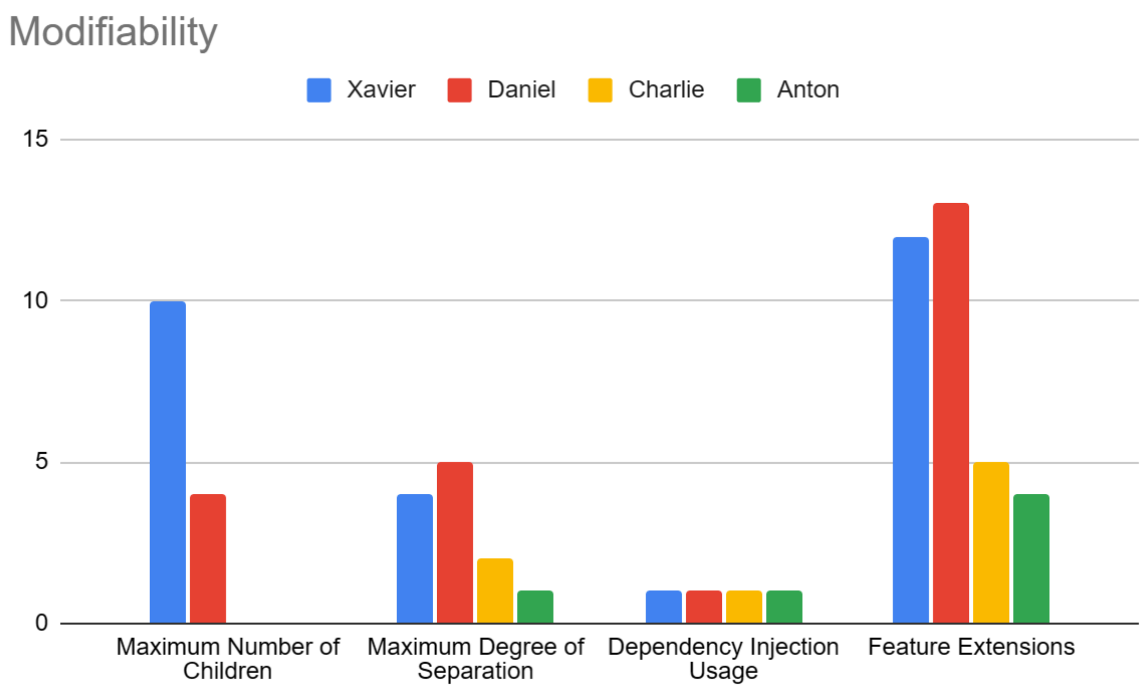
Based on our team’s prior understanding of Object-Oriented metrics we also devised another objective measure which was the Maximum Degree of Separation Between Classes.

Additionally, we also came up with 2 other subjective measures including a score out of 5 for usage of Dependency Injection and a score out of 20 for flexibility of extension based on the following possible extensions:

- Flexibility in varying number of segments per volcano card.
- Flexibility in adding new behaviour on Chit Card flip.
- Flexibility in varying number of players.

Which were decided based on our Sprint 1 documentation where we presented a number of possible additions in the form of user stories.

The results can be found in the Figure below.



Maintainability

Maintainability is the only provided major category in the IEEE spec and so covers a broad range of potential concepts that affect maintenance. Because of this we included 5 metrics:

A 0/1 scoring-based metric was used for the number of code smells detected in each codebase based on a selection of relevant code smells offered at www.refactoring.guru [3]. This was chosen as an accumulation of code smells over time is known to lead to increased technical debt.

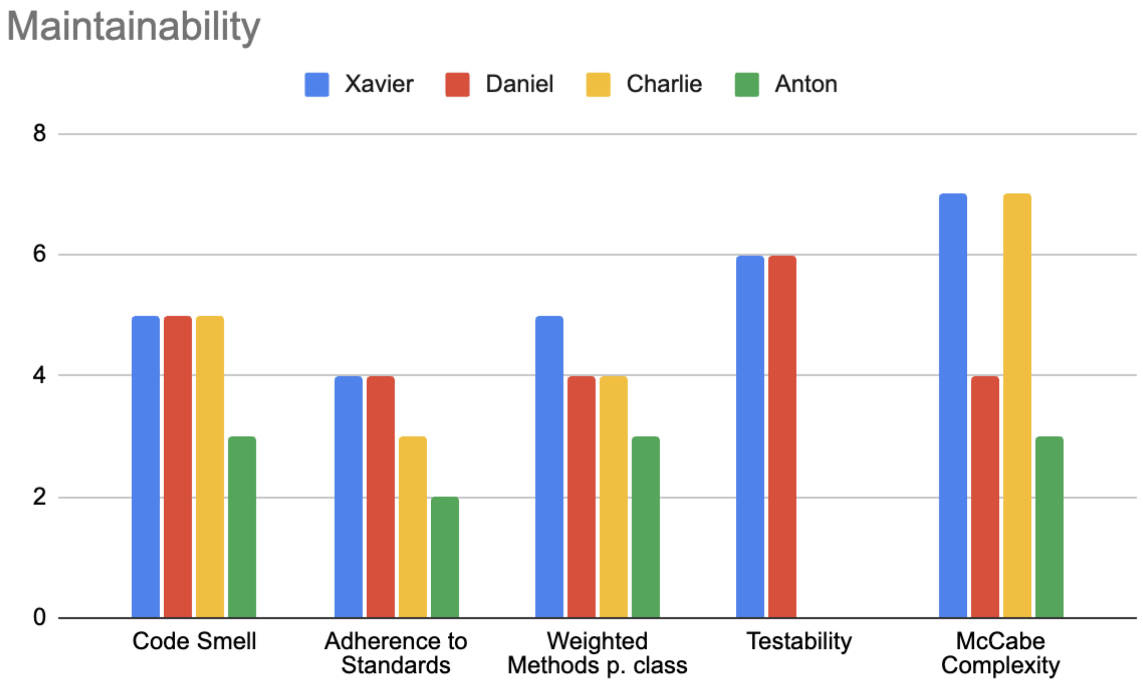
Adherence to coding standards was also assessed based on the clarity of the code's file structure, presence and effective use of type hints, and usage of doc strings. This metric was considered important because being able to understand a codebase more easily allows team members to work quickly while still making use of others' code. File structure and doc strings aid this understanding and with effective Language Server usage in one's Integrated Development Environment (IDE) typing can greatly speed up many programming tasks.

We also referenced the Chidamber and Kemerer article [2] again by considering the "Weighted Methods per Class" and specifically weighting by the functionality domains expressed in the Sprint 2 brief.

A scoring-based metric was also assigned to Test Usage and Testability where we considered factors such as how testable functions were without interfacing with the pygame GUI (minimises slow manual tests), percentage of the code tested, and a separate score for how testable each of the key interactions were based on the primary involved methods.

Finally, we made use of a Python linting tool to measure the Cyclomatic/McCabe Complexity of our code. This value tends to indicate overly complex methods and thus difficulty in maintaining the code when any changes need to be made.

The results of all these metrics are available in the below Figure.

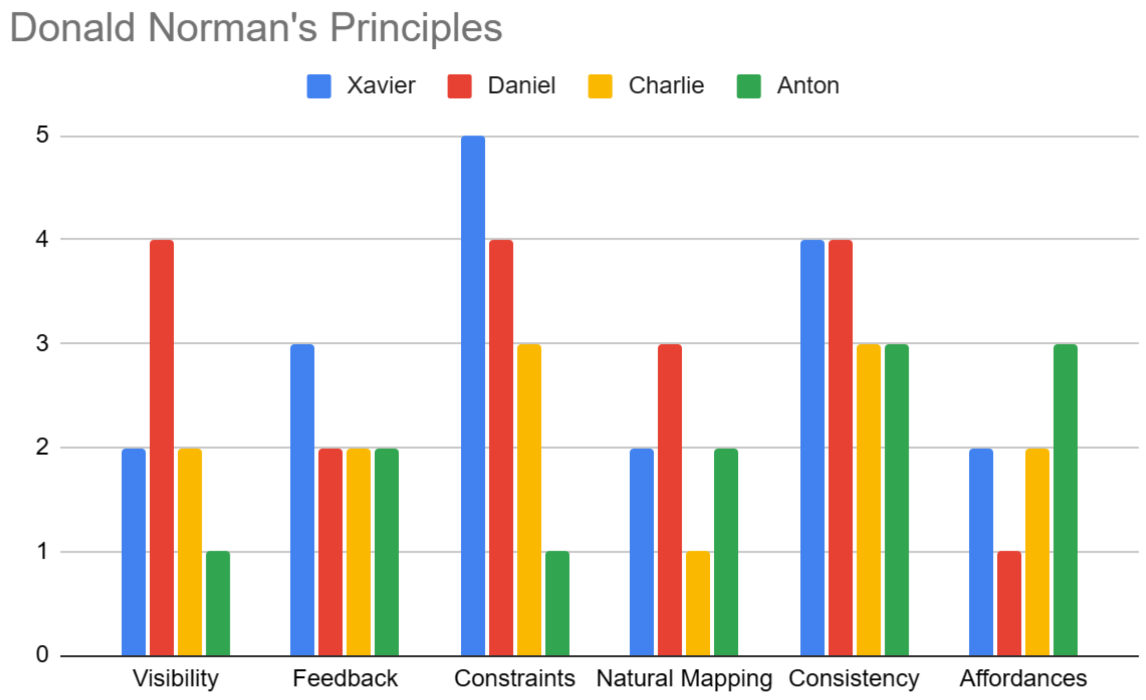


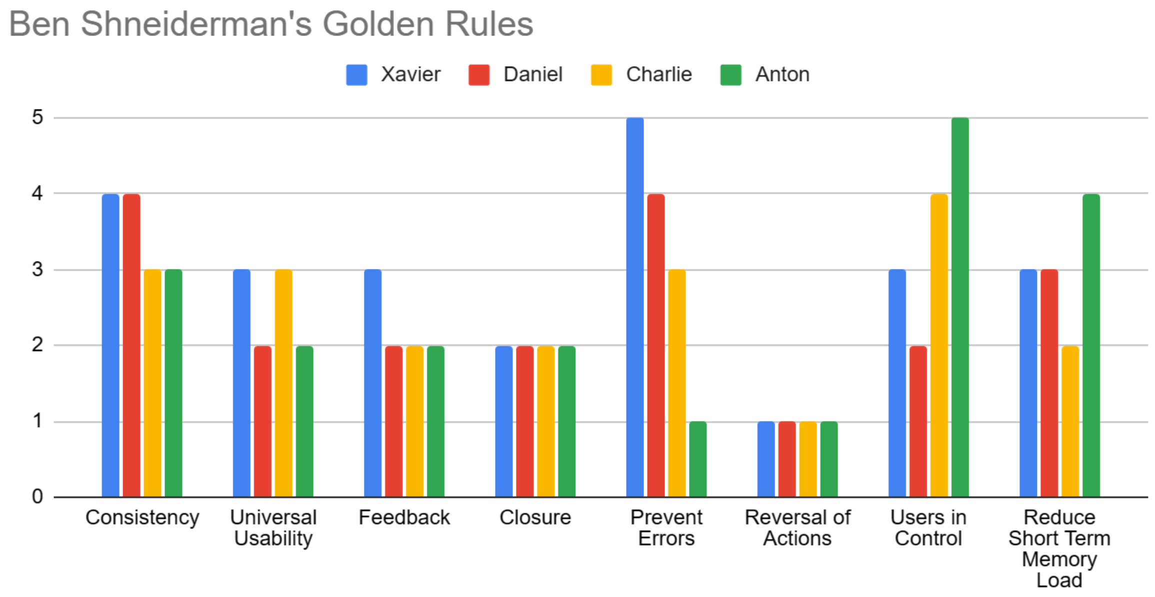
User engagement

User engagement is again a highly subjective measure and so precise metrics are not reasonable in this case. The team decided to make observations made at game runtime along with consideration of Donald Norman's Design Principles [4] and Ben Shneiderman's 8 Golden Rules [5], two of the most highly regarded principle sets in design, to assess each game.

We also incorporated a rating for "Game Feel" which is a common term used in Game Development to express the value of features such as on-hover effects on buttons, special effects, tactile gameplay, etc. However, with each game being only demos these features were not included in any game resulting in a minimum score across all codebases. However, we have chosen to still consider this as it should drive our future decisions in Sprint 3 and 4.

The results are given in the Figure below.





Key Findings

One of the key details of note is that across-the-board Xavier and Daniel's Sprint 2 designs and codebases scored consistently high across many metrics in different categories. This potentially indicates that for Sprint 3 one of these Sprints could be.

Throughout the review it was noted that each Sprint 2 branch had consistent features that effectively scored or lost points. One of these features of Xavier's design and implementation was the successful component system which was considered as part of Sprint 1. This feature in particular was considered beneficial for certain maintainability and modifiability metrics such as avoiding long methods, avoiding God classes, and adding greater flexibility. As a result, we will strongly consider taking this branch as a base going into Sprint 3.

It was also noted through the exploration of each of the codebases that any documentation and typing was already helping significantly with the analysis of the code and as a result in-code documentation and type hinting should be placed as a goal during Sprint 3.

When considering Design Pattern usage, we recognised that the Observer pattern and the Command pattern were among the most appropriately implemented so these will be highly considered for usage going forward. However, the specific implementation of the Observer pattern in Daniel's Sprint 2 branch relied on a switch statement for typing casting which is antithetical to the Liskov Substitution Principle and so may need to be rethought. Additionally, the Singleton design saw use during Sprint 2. In general, the uses of this pattern did not stand out to us as overly effective, however due to many game objects/components often needing access to pieces of global state it is likely this pattern will see use in Sprint 3.

Finally, the results of user engagement evaluation were quite varied across the board. It is clear that in Sprint 3 we should work to achieve visual consistency as well as address some of the key aspects of the 2 different principle sets that were seen to have low scores in Sprint 2 such as Visibility, Affordances, Reversal of Actions, Closure and Feedback. It should also be noted that effort in ensuring high user engagement should be expended across all facets of the game's implementation as even small discrepancies may significantly degrade user experience.

Object-Oriented Design

Class Responsibility Collaborator Cards

ChitCard	
● Know if it is showing or hidden ● Store its animal type and amount ● When clicked flip over and display its animal type and count ● Flip over once the players turn has finished	● PlayerTurnEndEmitter

Alternatives:

- ChitCards, being Undateables, could actively check for the status of the turn without having to collaborate with the PlayerTurnEmitter. Rejected as the alternative would require even more dependencies and would potentially produce future modifiability constraints e.g. changes to how active Player is stored.

Player	
● Know which player's turn it is ● Know the path that it has to take ● Know which segment it is on ● Move itself to a new segment ● Determine if it should or shouldn't move based on last flipped chit card	● Segment

Alternatives:

- A GameBoard singleton could be used to hold important global state such as who the active Player is. Rejected as this opens up the possibility of either a pure dataclass for only storing this state or a god class with too many responsibilities.
- The Player could compute movement on each ChitCard flipped but this was rejected based on Sprint 2 experience. It was found that computing the static path ahead of time simplified implementation.

Segment	
● Store its animal type ● Store the position that players move to when landing on this segment ● Know if there is a cave attached to itself ● Store the next segment ● Generate a path from this segment around the circle and back	● AnimalType ● TransformComponent

Alternatives:

- Players could perform all path calculations by simply using Segments as traversable nodes. This was rejected as it needlessly reduces active collaboration with Segments and essentially reduces them to dataclasses with very little functionality.

RenderableComponent	
● Store its colour ● Store its transform ● Know if it is visible ● Know what image it draws to a surface ● Rotate scale and offset images based on transform component ● Draw said image on request	● TransformComponent ● Renderable

Alternatives:

- Colour could be omitted from this abstract component as not all renderables will strictly be able to make use of it (i.e. SpriteComponent). However, this causes difficulties when trying to make use of the Liskov Substitution Principle in implementation.

SetColourCommand	
● Store the renderable component to change colour of ● Store the colour to change the renderable component to ● When told to run change the colour of the stored renderable component to its stored colour	● RenderableComponent ● Colour ● Command

Alternatives:

- Concrete classes could set the colour on the renderable component directly. This is antithetical to our design which utilises commands in order to modify the game state of other objects. It allows modification of the set colour command to include other features such as colour blur or delay animations without modifying the renderable component class improving separation of domains

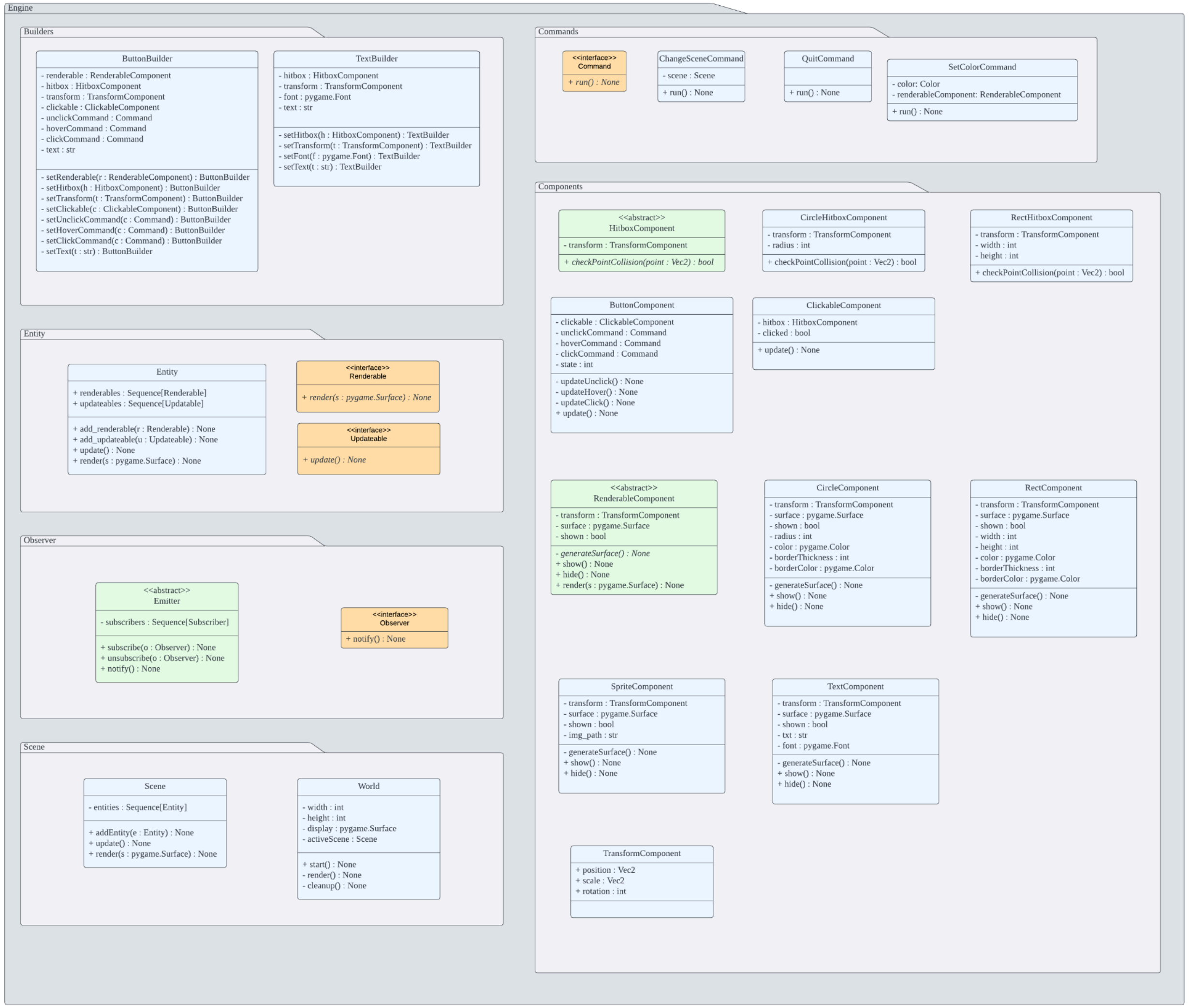
Entity	
● Store the components that make up an individual game object ● Pass on update and renderable commands to its components	● Updatable ● Renderable ● Scene

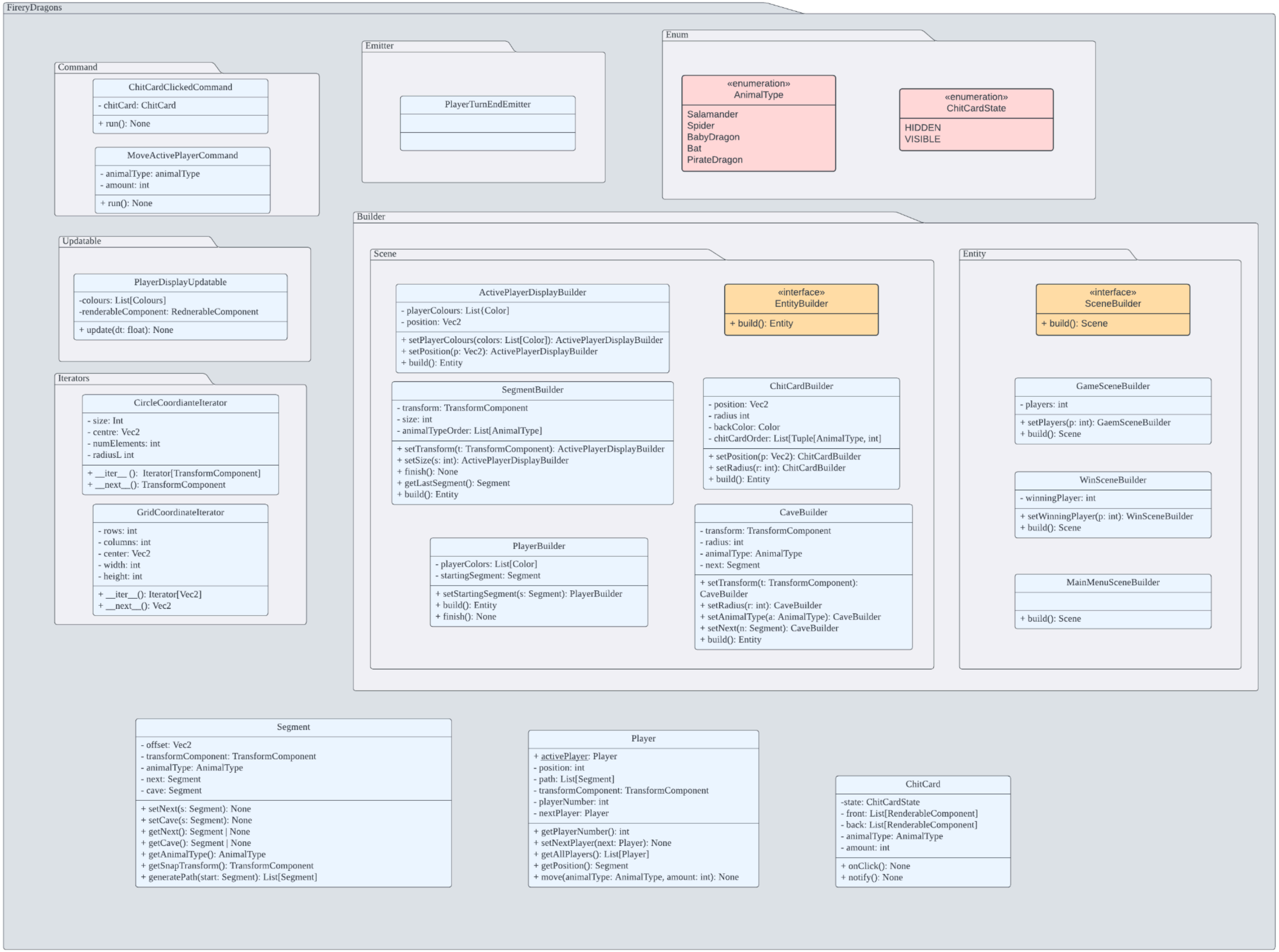
Alternatives:

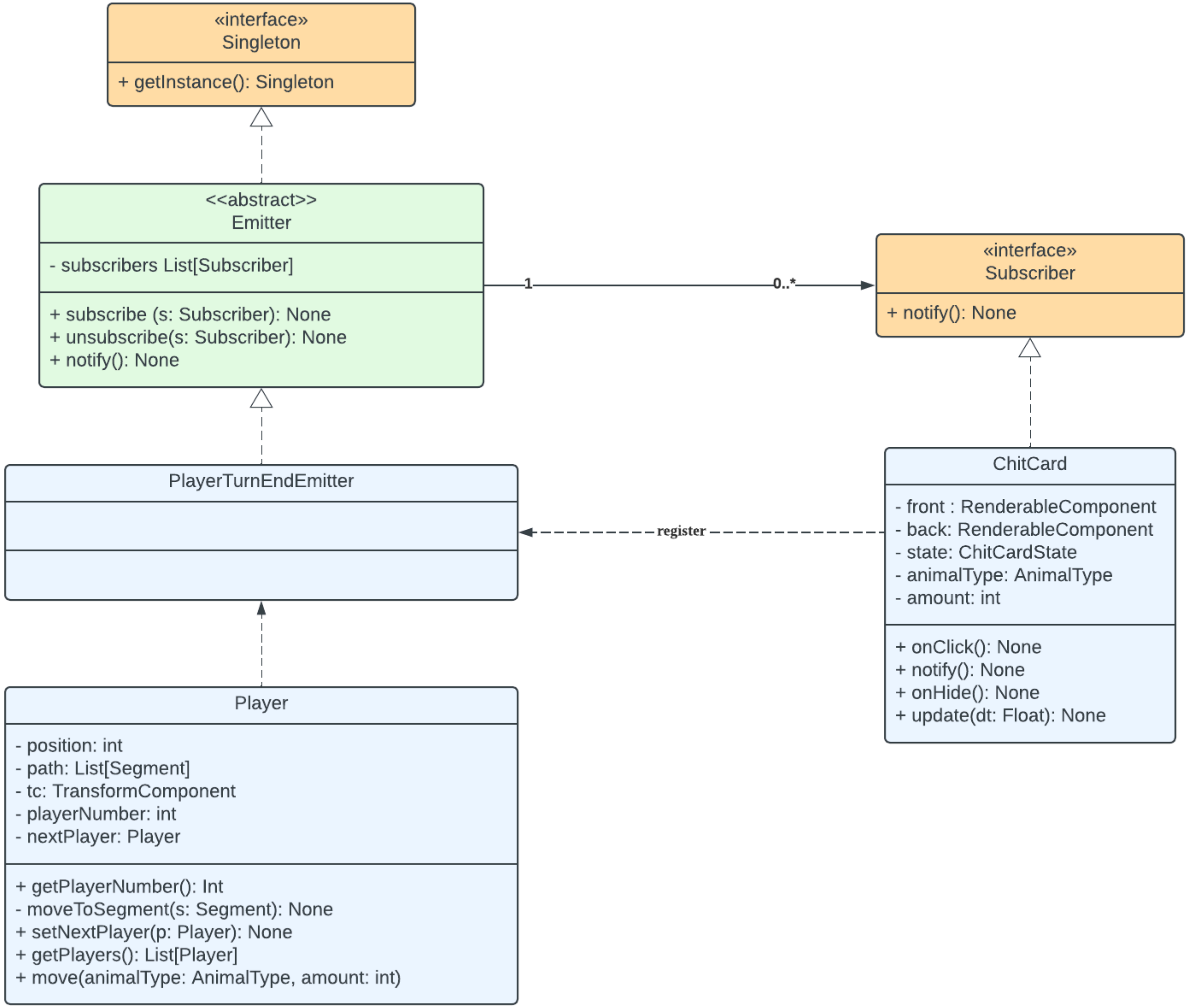
- Concrete classes such as ChitCard, Segment and Player could exist as the primary game entities instead of a general Entity class. This design however opens up opportunities for greater modularity which will help greatly in Sprint 4.

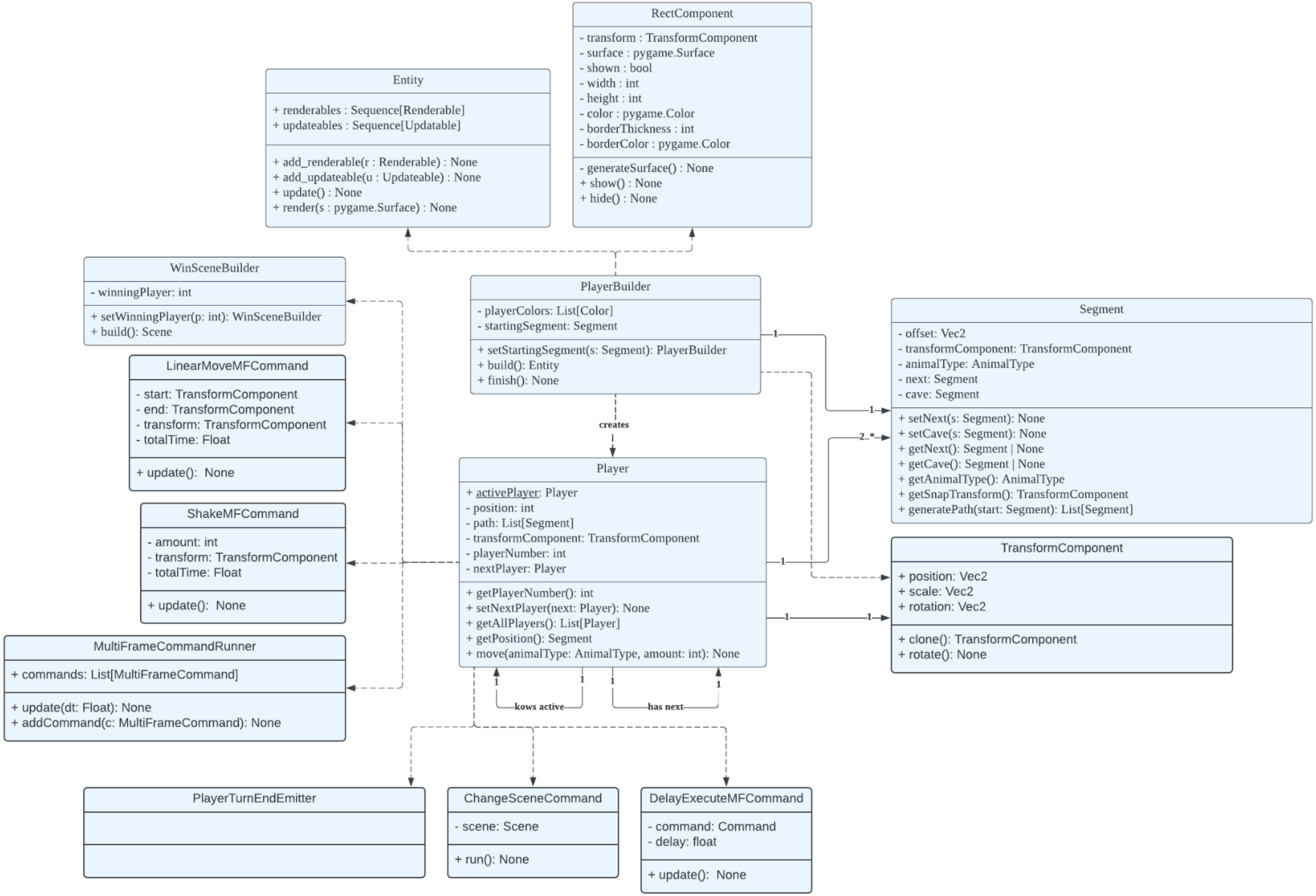
UML Class Diagram

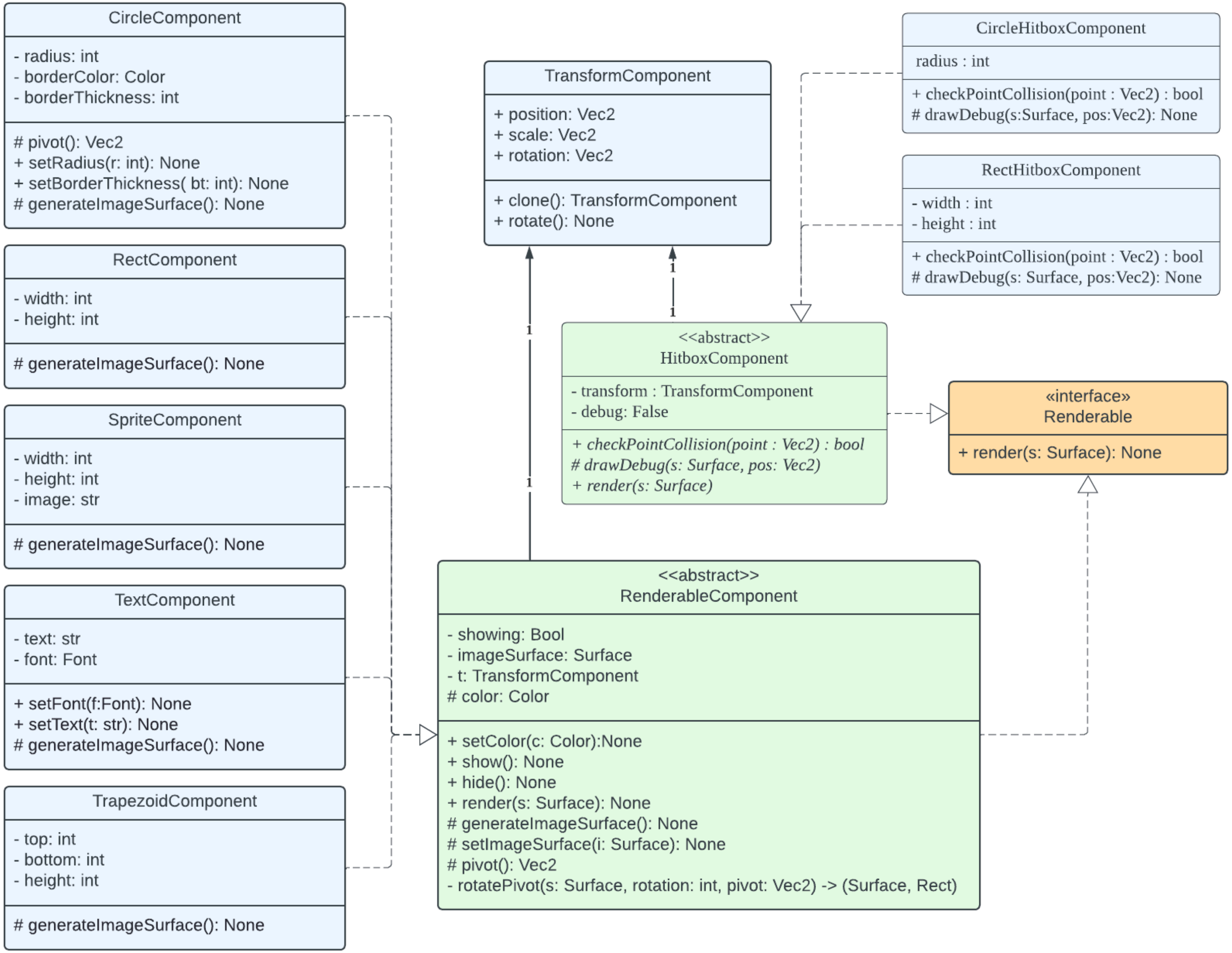
Pictured below are two UML diagram representing the design of the entire project in the Engine package and FieryDragons package. For readability purposes this UML focuses exclusively on the structure and individual class design. Also included are more focused diagrams that include the relations and dependencies between significant groups of classes that represent key functionality.

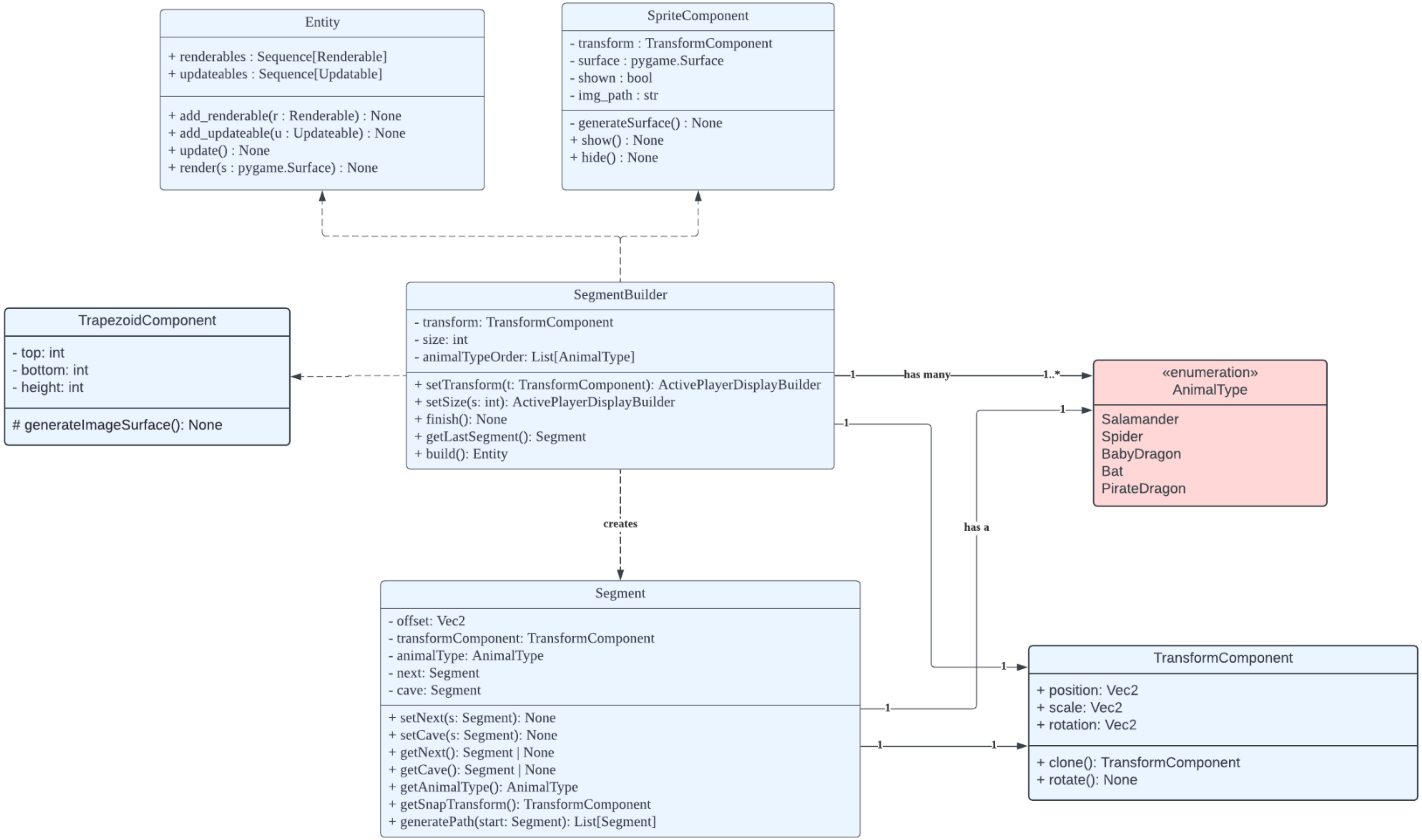

Engine
Builders
ButtonBuilder
- renderable : RenderableComponent
- hitbox : HitboxComponent
- transform : TransformComponent
- clickable : ClickableComponent
- unclickCommand : Command
- hoverCommand : Command
- clickCommand : Command
- text : str
- setRenderable(r : RenderableComponent) : ButtonBuilder
- setHitbox(h : HitboxComponent) : ButtonBuilder
- setTransform(t : TransformComponent) : ButtonBuilder
- setClickable(c : ClickableComponent) : ButtonBuilder
- setUnclickCommand(c : Command) : ButtonBuilder
- setHoverCommand(c : Command) : ButtonBuilder
- setClickCommand(c : Command) : ButtonBuilder
- setText(t : str) : ButtonBuilder
TextBuilder
- hitbox : HitboxComponent
- transform : TransformComponent
- font : pygame.Font
- text : str
- setHitbox(h : HitboxComponent) : TextBuilder
- setTransform(t : TransformComponent) : TextBuilder
- setFont(f : pygame.Font) : TextBuilder
- setText(t : str) : TextBuilder
Entity
Entity
+ renderables : Sequence[Renderable]
+ updateables : Sequence[Updatable]
+ add_renderable(r : Renderable) : None
+ add_updateable(u : Updateable) : None
+ update() : None
+ render(s : pygame.Surface) : None
<<interface>>
Renderable
+ render(s : pygame.Surface) : None
<<interface>>
Updateable
+ update() : None
Observer
<<abstract>>
Emitter
- subscribers : Sequence[Subscriber]
+ subscribe(o : Observer) : None
+ unsubscribe(o : Observer) : None
+ notify() : None
<<interface>>
Observer
+ notify() : None
Scene
Scene
- entities : Sequence[Entity]
+ addEntity(e : Entity) : None
+ update() : None
+ render(s : pygame.Surface) : None
World
- width : int
- height : int
- display : pygame.Surface
- activeScene : Scene
+ start() : None
- render() : None
- cleanup() : None
Commands
<<interface>>
Command
+ run() : None
ChangeSceneCommand
- scene : Scene
+ run() : None
QuitCommand
+ run() : None
SetColorCommand
- color: Color
- renderableComponent: RenderableComponent
+ run() : None
Components
<<abstract>>
HitboxComponent
- transform : TransformComponent
+ checkPointCollision(point : Vec2) : bool
CircleHitboxComponent
- transform : TransformComponent
- radius : int
+ checkPointCollision(point : Vec2) : bool
RectHitboxComponent
- transform : TransformComponent
- width : int
- height : int
+ checkPointCollision(point : Vec2) : bool
ButtonComponent
- clickable : ClickableComponent
- unclickCommand : Command
- hoverCommand : Command
- clickCommand : Command
- state : int
- updateUnclick() : None
- updateHover() : None
- updateClick() : None
+ update() : None
ClickableComponent
- hitbox : HitboxComponent
- clicked : bool
+ update() : None
<<abstract>>
RenderableComponent
- transform : TransformComponent
- surface : pygame.Surface
- shown : bool
- generateSurface() : None
+ show() : None
+ hide() : None
+ render(s : pygame.Surface) : None
CircleComponent
- transform : TransformComponent
- surface : pygame.Surface
- shown : bool
- radius : int
- color : pygame.Color
- borderThickness : int
- borderColor : pygame.Color
- generateSurface() : None
+ show() : None
+ hide() : None
RectComponent
- transform : TransformComponent
- surface : pygame.Surface
- shown : bool
- width : int
- height : int
- color : pygame.Color
- borderThickness : int
- borderColor : pygame.Color
- generateSurface() : None
+ show() : None
+ hide() : None
SpriteComponent
- transform : TransformComponent
- surface : pygame.Surface
- shown : bool
- img_path : str
- generateSurface() : None
+ show() : None
+ hide() : None
TextComponent
- transform : TransformComponent
- surface : pygame.Surface
- shown : bool
- txt : str
- font : pygame.Font
- generateSurface() : None
+ show() : None
+ hide() : None
TransformComponent
+ position : Vec2
+ scale : Vec2
+ rotation : int

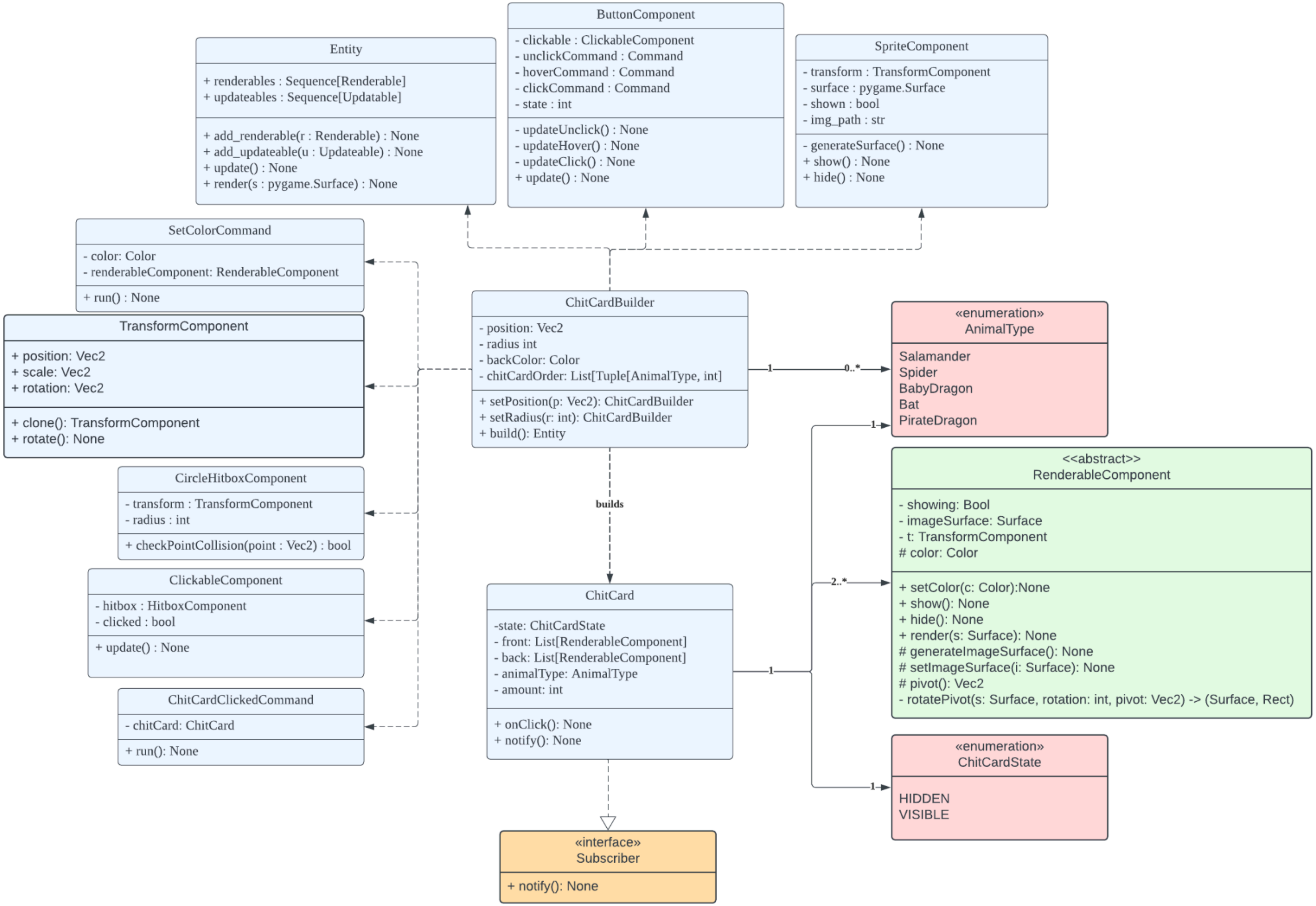

FireryDragons
Command
ChitCardClickedCommand
- chitCard: ChitCard
+ run(): None
MoveActivePlayerCommand
- animalType: animalType
- amount: int
+ run(): None
Emitter
PlayerTurnEndEmitter
Enum
«enumeration»
AnimalType
Salamander
Spider
BabyDragon
Bat
PirateDragon
«enumeration»
ChitCardState
HIDDEN
VISIBLE
Updatable
PlayerDisplayUpdatable
-colours: List[Colours]
-renderableComponent: RednerableComponent
+ update(dt: float): None
Iterators
CircleCoordianteIterator
- size: Int
- centre: Vec2
- numElements: int
- radiusL int
+ __iter__ (): Iterator[TransformComponent]
+ __next__(): TransformComponent
GridCoordinateIterator
- rows: int
- columns: int
- center: Vec2
- width: int
- height: int
+ __iter__(): Iterator[Vec2]
+ __next__(): Vec2
Builder
Scene
ActivePlayerDisplayBuilder
- playerColours: List{Color]
- position: Vec2
+ setPlayerColours(colors: List[Color]): ActivePlayerDisplayBuilder
+ setPosition(p: Vec2): ActivePlayerDisplayBuilder
+ build(): Entity
«interface»
EntityBuilder
+ build(): Entity
SegmentBuilder
- transform: TransformComponent
- size: int
- animalTypeOrder: List[AnimalType]
+ setTransform(t: TransformComponent): ActivePlayerDisplayBuilder
+ setSize(s: int): ActivePlayerDisplayBuilder
+ finish(): None
+ getLastSegment(): Segment
+ build(): Entity
ChitCardBuilder
- position: Vec2
- radius int
- backColor: Color
- chitCardOrder: List[Tuple[AnimalType, int]
+ setPosition(p: Vec2): ChitCardBuilder
+ setRadius(r: int): ChitCardBuilder
+ build(): Entity
PlayerBuilder
- playerColors: List[Color]
- startingSegment: Segment
+ setStartingSegment(s: Segment): PlayerBuilder
+ build(): Entity
+ finish(): None
CaveBuilder
- transform: TransformComponent
- radius: int
- animalType: AnimalType
- next: Segment
+ setTransform(t: TransformComponent): CaveBuilder
+ setRadius(r: int): CaveBuilder
+ setAnimalType(a: AnimalType): CaveBuilder
+ setNext(n: Segment): CaveBuilder
+ build(): Entity
Entity
«interface»
SceneBuilder
+ build(): Scene
GameSceneBuilder
- players: int
+ setPlayers(p: int): GaemSceneBuilder
+ build(): Scene
WinSceneBuilder
- winningPlayer: int
+ setWinningPlayer(p: int): WinSceneBuilder
+ build(): Scene
MainMenuSceneBuilder
+ build(): Scene
Segment
- offset: Vec2
- transformComponent: TransformComponent
- animalType: AnimalType
- next: Segment
- cave: Segment
+ setNext(s: Segment): None
+ setCave(s: Segment): None
+ getNext(): Segment | None
+ getCave(): Segment | None
+ getAnimalType(): AnimalType
+ getSnapTransform(): TransformComponent
+ generatePath(start: Segment): List[Segment]
Player
+ activePlayer: Player
- position: int
- path: List[Segment]
- transformComponent: TransformComponent
- playerNumber: int
- nextPlayer: Player
+ getPlayerNumber(): int
+ setNextPlayer(next: Player): None
+ getAllPlayers(): List[Player]
+ getPosition(): Segment
+ move(animalType: AnimalType, amount: int): None
ChitCard
-state: ChitCardState
- front: List[RenderableComponent]
- back: List[RenderableComponent]
- animalType: AnimalType
- amount: int
+ onClick(): None
+ notify(): None


«interface»
Singleton
+ getInstance(): Singleton
<<abstract>>
Emitter
- subscribers List[Subscriber]
+ subscribe (s: Subscriber): None
+ unsubscribe(s: Subscriber): None
+ notify(): None
1
0..*
«interface»
Subscriber
+ notify(): None
PlayerTurnEndEmitter
register
ChitCard
- front : RenderableComponent
- back: RenderableComponent
- state: ChitCardState
- animalType: AnimalType
- amount: int
+ onClick(): None
+ notify(): None
+ onHide(): None
+ update(dt: Float): None
Player
- position: int
- path: List[Segment]
- tc: TransformComponent
- playerNumber: int
- nextPlayer: Player
+ getPlayerNumber(): Int
- moveToSegment(s: Segment): None
+ setNextPlayer(p: Player): None
+ getPlayers(): List[Player]
+ move(animalType: AnimalType, amount: int)


RectComponent
- transform : TransformComponent
- surface : pygame.Surface
- shown : bool
- width : int
- height : int
- color : pygame.Color
- borderThickness : int
- borderColor : pygame.Color
- generateSurface() : None
+ show() : None
+ hide() : None
Entity
+ renderables : Sequence[Renderable]
+ updateables : Sequence[Updatable]
+ add_renderable(r : Renderable) : None
+ add_updateable(u : Updateable) : None
+ update() : None
+ render(s : pygame.Surface) : None
WinSceneBuilder
- winningPlayer: int
+ setWinningPlayer(p: int): WinSceneBuilder
+ build(): Scene
PlayerBuilder
- playerColors: List[Color]
- startingSegment: Segment
+ setStartingSegment(s: Segment): PlayerBuilder
+ build(): Entity
+ finish(): None
Segment
- offset: Vec2
- transformComponent: TransformComponent
- animalType: AnimalType
- next: Segment
- cave: Segment
+ setNext(s: Segment): None
+ setCave(s: Segment): None
+ getNext(): Segment | None
+ getCave(): Segment | None
+ getAnimalType(): AnimalType
+ getSnapTransform(): TransformComponent
+ generatePath(start: Segment): List[Segment]
1
1
2..*
LinearMoveMFCommand
- start: TransformComponent
- end: TransformComponent
- transform: TransformComponent
- totalTime: Float
+ update(): None
creates
Player
+ activePlayer: Player
- position: int
- path: List[Segment]
- transformComponent: TransformComponent
- playerNumber: int
- nextPlayer: Player
+ getPlayerNumber(): int
+ setNextPlayer(next: Player): None
+ getAllPlayers(): List[Player]
+ getPosition(): Segment
+ move(animalType: AnimalType, amount: int): None
1
1
1
ShakeMFCommand
- amount: int
- transform: TransformComponent
- totalTime: Float
+ update(): None
TransformComponent
+ position: Vec2
+ scale: Vec2
+ rotation: Vec2
+ clone(): TransformComponent
+ rotate(): None
MultiFrameCommandRunner
+ commands: List[MultiFrameCommand]
+ update(dt: Float): None
+ addCommand(c: MultiFrameCommand): None
1
1
kows active
1
1
has next
PlayerTurnEndEmitter
ChangeSceneCommand
- scene: Scene
+ run(): None
DelayExecuteMFCommand
- command: Command
- delay: float
+ update(): None


CircleComponent
- radius: int
- borderColor: Color
- borderThickness: int
# pivot(): Vec2
+ setRadius(r: int): None
+ setBorderThickness(bt: int): None
# generateImageSurface(): None
RectComponent
- width: int
- height: int
# generateImageSurface(): None
SpriteComponent
- width: int
- height: int
- image: str
# generateImageSurface(): None
TextComponent
- text: str
- font: Font
+ setFont(f:Font): None
+ setText(t: str): None
# generateImageSurface(): None
TrapezoidComponent
- top: int
- bottom: int
- height: int
# generateImageSurface(): None
TransformComponent
+ position: Vec2
+ scale: Vec2
+ rotation: Vec2
+ clone(): TransformComponent
+ rotate(): None
CircleHitboxComponent
radius : int
+ checkPointCollision(point : Vec2) : bool
# drawDebug(s:Surface, pos:Vec2): None
RectHitboxComponent
- width : int
- height : int
+ checkPointCollision(point : Vec2) : bool
# drawDebug(s: Surface, pos:Vec2): None
1
1
1
1
<<abstract>>
HitboxComponent
- transform : TransformComponent
- debug: False
+ checkPointCollision(point : Vec2) : bool
# drawDebug(s: Surface, pos: Vec2)
+ render(s: Surface)
«interface»
Renderable
+ render(s: Surface): None
<<abstract>>
RenderableComponent
- showing: Bool
- imageSurface: Surface
- t: TransformComponent
# color: Color
+ setColor(c: Color):None
+ show(): None
+ hide(): None
+ render(s: Surface): None
# generateImageSurface(): None
# setImageSurface(i: Surface): None
# pivot(): Vec2
- rotatePivot(s: Surface, rotation: int, pivot: Vec2) -> (Surface, Rect)


Entity
+ renderables : Sequence[Renderable]
+ updateables : Sequence[Updatable]
+ add_renderable(r : Renderable) : None
+ add_updateable(u : Updateable) : None
+ update() : None
+ render(s : pygame.Surface) : None
SpriteComponent
- transform : TransformComponent
- surface : pygame.Surface
- shown : bool
- img_path : str
- generateSurface() : None
+ show() : None
+ hide() : None
TrapezoidComponent
- top: int
- bottom: int
- height: int
# generateImageSurface(): None
SegmentBuilder
- transform: TransformComponent
- size: int
- animalTypeOrder: List[AnimalType]
+ setTransform(t: TransformComponent): ActivePlayerDisplayBuilder
+ setSize(s: int): ActivePlayerDisplayBuilder
+ finish(): None
+ getLastSegment(): Segment
+ build(): Entity
1
has many
1..*
«enumeration»
AnimalType
Salamander
Spider
BabyDragon
Bat
PirateDragon
1
1
creates
has a
Segment
- offset: Vec2
- transformComponent: TransformComponent
- animalType: AnimalType
- next: Segment
- cave: Segment
+ setNext(s: Segment): None
+ setCave(s: Segment): None
+ getNext(): Segment | None
+ getCave(): Segment | None
+ getAnimalType(): AnimalType
+ getSnapTransform(): TransformComponent
+ generatePath(start: Segment): List[Segment]
1
1
1
1
TransformComponent
+ position: Vec2
+ scale: Vec2
+ rotation: Vec2
+ clone(): TransformComponent
+ rotate(): None



Entity

+ renderables : Sequence[Renderable]
+ updateables : Sequence[Updatable]

+ add_renderable(r : Renderable) : None
+ add_updateable(u : Updateable) : None
+ update() : None
+ render(s : pygame.Surface) : None

ButtonComponent

- clickable : ClickableComponent
- unclickCommand : Command
- hoverCommand : Command
- clickCommand : Command
- state : int

- updateUnclick() : None
- updateHover() : None
- updateClick() : None
+ update() : None

SpriteComponent

- transform : TransformComponent
- surface : pygame.Surface
- shown : bool
- img_path : str

- generateSurface() : None
+ show() : None
+ hide() : None

SetColorCommand

- color: Color
- renderableComponent: RenderableComponent

+ run() : None

TransformComponent

+ position: Vec2
+ scale: Vec2
+ rotation: Vec2

+ clone(): TransformComponent
+ rotate(): None

CircleHitboxComponent

- transform : TransformComponent
- radius : int

+ checkPointCollision(point : Vec2) : bool

ClickableComponent

- hitbox : HitboxComponent
- clicked : bool

+ update() : None

ChitCardClickedCommand

- chitCard: ChitCard

+ run(): None

ChitCardBuilder

- position: Vec2
- radius int
- backColor: Color
- chitCardOrder: List[Tuple[AnimalType, int]

+ setPosition(p: Vec2): ChitCardBuilder
+ setRadius(r: int): ChitCardBuilder
+ build(): Entity

ChitCard

-state: ChitCardState
- front: List[RenderableComponent]
- back: List[RenderableComponent]
- animalType: AnimalType
- amount: int

+ onClick(): None
+ notify(): None

«interface»
Subscriber

+ notify(): None

«enumeration»
AnimalType

Salamander
Spider
BabyDragon
Bat
PirateDragon

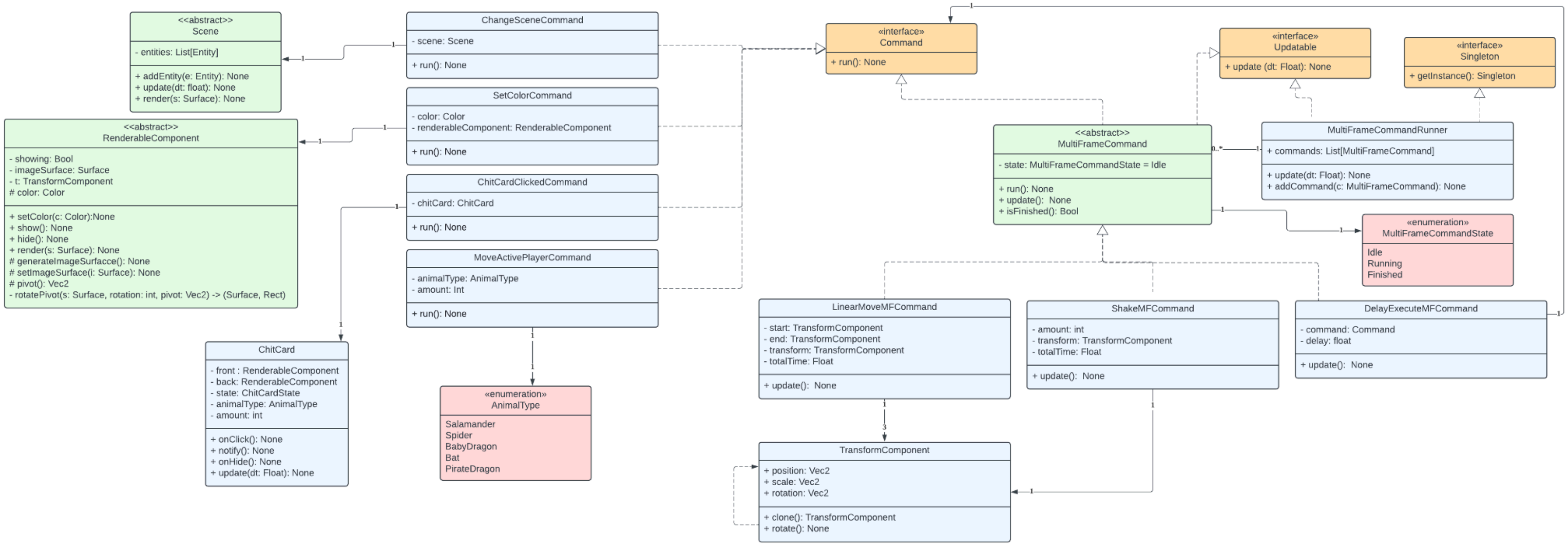
<<abstract>>
RenderableComponent

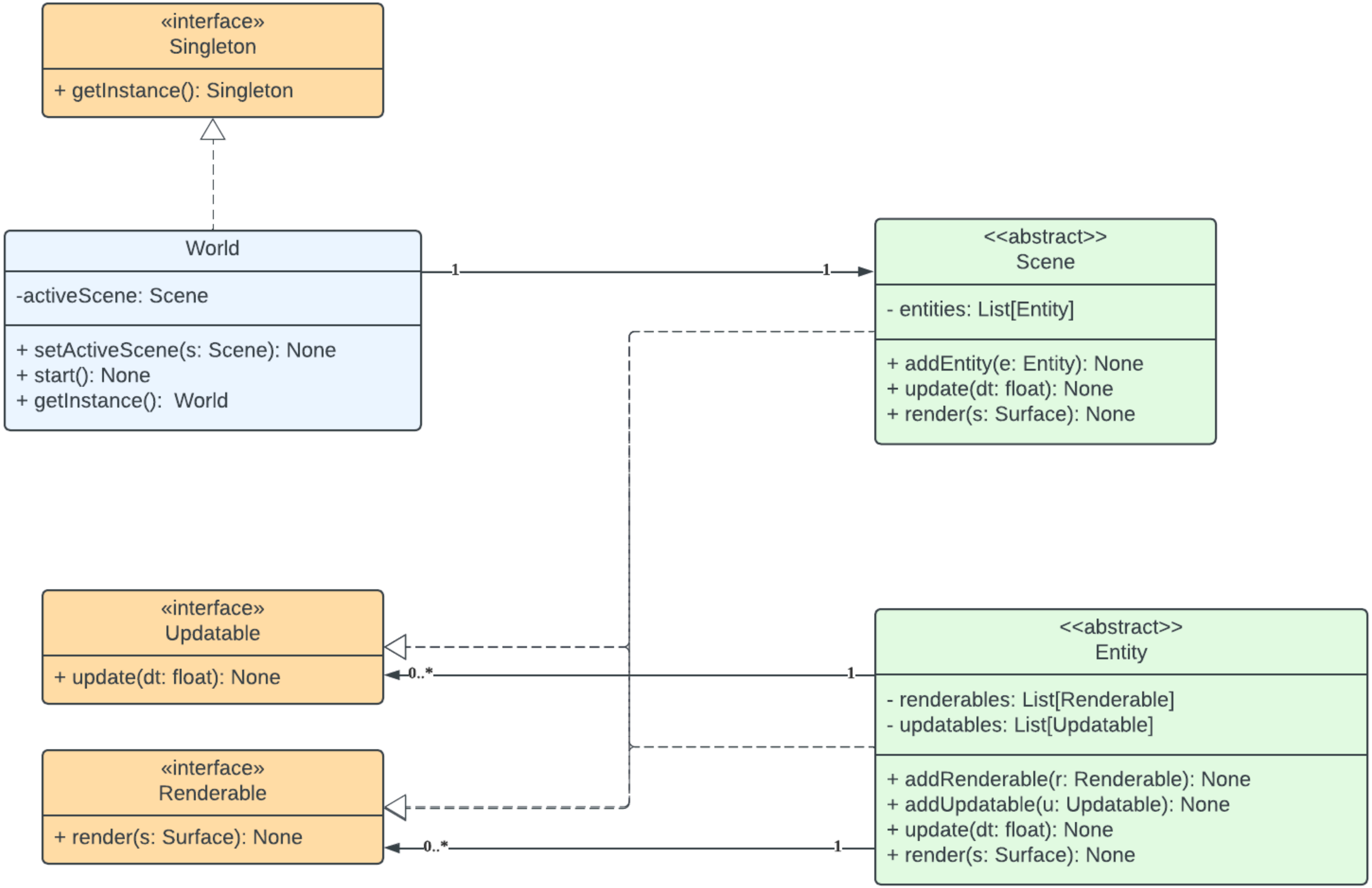
- showing: Bool
- imageSurface: Surface
- t: TransformComponent
color: Color

+ setColor(c: Color):None
+ show(): None
+ hide(): None
+ render(s: Surface): None
# generateImageSurface(): None
# setImageSurface(i: Surface): None
# pivot(): Vec2
- rotatePivot(s: Surface, rotation: int, pivot: Vec2) -> (Surface, Rect)

«enumeration»
ChitCardState

HIDDEN
VISIBLE


<<abstract>>
Scene
- entities: List[Entity]
+ addEntity(e: Entity): None
+ update(dt: float): None
+ render(s: Surface): None
<<abstract>>
RenderableComponent
- showing: Bool
- imageSurface: Surface
- t: TransformComponent
# color: Color
+ setColor(c: Color):None
+ show(): None
+ hide(): None
+ render(s: Surface): None
# generateImageSurfacce(): None
# setImageSurface(i: Surface): None
# pivot(): Vec2
- rotatePivot(s: Surface, rotation: int, pivot: Vec2) -> (Surface, Rect)
ChangeSceneCommand
- scene: Scene
+ run(): None
SetColorCommand
- color: Color
- renderableComponent: RenderableComponent
+ run(): None
ChitCardClickedCommand
- chitCard: ChitCard
+ run(): None
MoveActivePlayerCommand
- animalType: AnimalType
- amount: Int
+ run(): None
ChitCard
- front : RenderableComponent
- back: RenderableComponent
- state: ChitCardState
- animalType: AnimalType
- amount: int
+ onClick(): None
+ notify(): None
+ onHide(): None
+ update(dt: Float): None
«enumeration»
AnimalType
Salamander
Spider
BabyDragon
Bat
PirateDragon
«interface»
Command
+ run(): None
«interface»
Updatable
+ update (dt: Float): None
«interface»
Singleton
+ getInstance(): Singleton
<<abstract>>
MultiFrameCommand
- state: MultiFrameCommandState = Idle
+ run(): None
+ update(): None
+ isFinished(): Bool
MultiFrameCommandRunner
+ commands: List[MultiFrameCommand]
+ update(dt: Float): None
+ addCommand(c: MultiFrameCommand): None
«enumeration»
MultiFrameCommandState
Idle
Running
Finished
LinearMoveMFCommand
- start: TransformComponent
- end: TransformComponent
- transform: TransformComponent
- totalTime: Float
+ update(): None
ShakeMFCommand
- amount: int
- transform: TransformComponent
- totalTime: Float
+ update(): None
DelayExecuteMFCommand
- command: Command
- delay: float
+ update(): None
TransformComponent
+ position: Vec2
+ scale: Vec2
+ rotation: Vec2
+ clone(): TransformComponent
+ rotate(): None
1
1
1
1
1
1
1
1
1
0..*
1
1
1
1
3
1
1
1
1


«interface»
Singleton
+ getInstance(): Singleton
World
-activeScene: Scene
+ setActiveScene(s: Scene): None
+ start(): None
+ getInstance(): World
1
1
<<abstract>>
Scene
- entities: List[Entity]
+ addEntity(e: Entity): None
+ update(dt: float): None
+ render(s: Surface): None
«interface»
Updatable
+ update(dt: float): None
0..*
1
«interface»
Renderable
+ render(s: Surface): None
0..*
1
<<abstract>>
Entity
- renderables: List[Renderable]
- updatables: List[Updatable]
+ addRenderable(r: Renderable): None
+ addUpdatable(u: Updatable): None
+ update(dt: float): None
+ render(s: Surface): None

Executable Information

Antivirus Issues

If the exe is flagged by windows defender or your antivirus then you must add it to its whitelist folder. For windows defender (windows 11):

1. Open windows security app
2. Navigate to virus and thread protection.
3. Click manage settings under Virus and Threat protection settings.
4. Scroll to Exclusions (at the bottom)
5. Click add or remove exclusions
6. Add the dist folder

Build

In the base directory (the location of the file build.bat), run the following command:

```
./build.bat
```

The exe file is located in dist/FieryDragons.exe

Run

1. Check whitelist settings as described in antivirus issues
2. Download the .zip and extract or build the project
3. Open the dist folder and double click FieryDragons.exe to run

References

[1] “ISO/IEC 25010.” Institute of Electrical and Electronics Engineers, 2024. Accessed: May 14, 2024. [Online]. Available: https://iso25000.com/index.php/en/iso-25000-standards/iso-25010

[2] S. R. Chidamber and C. F. Kemerer, “A metrics suite for object oriented design,” IEEE Trans. Softw. Eng., vol. 20, no. 6, pp. 476–493, Jun. 1994, doi: 10.1109/32.295895.

[3] A. Shvets, “Code Smells,” Refactoring. Accessed: May 14, 2024. [Online]. Available: https://refactoring.guru/refactoring/smells

[4] D. Norman, The design of everyday things. New York : Basic Books, 2013.

[5] B. Shneiderman, C. Plaisant, M. Cohen, S. Jacobs, N. Elmqvist, and N. Diakopoulos, Designing the User Interface, 6th ed. Harlow: Pearson Education, Limited, 2018.

Appendix A

Function Completeness and Correctness Subcategories are as follows:

Game Setup

- The board state should be randomised each time
- Correct number of Volcano Cards are created
- Each Volcano Card has the correct number of segments
- Volcano Cards & segments are presented in a ring structure
- Correct number of Players are created
- Players are positioned on their Caves
- There exists a Cave for each Player attached to 1 distinct Volcano Card
- Each Cave is positioned in the centre of a Volcano Card
- Correct number of Chit Cards are created
- Chit Cards are positioned in a grid inside the Volcano Card ring

Chit Card Flipping

- Chit Cards display face down and give no information on their frontside
- There is functionality to flip a specific Chit Card in place
- When a Chit Card is flipped there is an indicator of its Animal Type
- When a Chit Card is flipped there is an indicator of its count (one to three)

Dragon Token Movement

- There is functionality in place to receive data from a recently flipped Chit Card
- The Player only moves on their turn
- For non-pirate dragon flips the Player moves forward if the Animal Type data matches their position
- For pirate dragon flips the Player attempts to move backwards always
- The Player moves according to the count on the card
- The Player never moves backwards past their Cave (stop at Cave)
- The Player never moves forwards past their Cave (don't move)
- The Player never moves on top of another Player (don't move)
- The Player doesn't enter opponent Caves
- The Player enters their cave if possible
- Players move visually around the Volcano Card ring structure

Turn Changing

- Turn should end on failed movement
- Turn should move consistently from one Player to another (same order every time)
- Turn should "wrap around" when ending the "last Player's" turn

Winning the Game

- When a Player has successfully returned to their cave that Player should win
- Win condition should not trigger on backwards movement to Cave
- Game should end not allowing additional actions on Win
- It should be clear which Player has won the game (e.g. Win Screen)