

Titolo Documento	Autori	Specifiche
Documento di Analisi e Specifica dei Requisiti	Davide Lovat Samuele Lovat	Nome documento: Documento di Progettazione 1.0 Data avvio: 10/03/2014 Data rilascio: 17/03/2014 Data revisione: Versione: 1.0

Redatto da	Davide Lovat Samuele Lovat
Verificato da	Davide Lovat Samuele Lovat

Documento di Progettazione

Gruppo di Lavoro
Bad Mood

Sommario

Prefazione

1 Introduzione

- 1.1 Scopo del documento dei requisiti
- 1.2 Descrizione del resto del documento

2 Prima Applicazione Quiz Time

2.1 Progettazione Architetturale

- 2.1.1 Organizzazione del Sistema
- 2.1.2 Scomposizione Modulare
 - 2.1.2.1 Moduli dei Sottosistemi

2.2 Progettazione Orientata agli Oggetti

- 2.2.1 Diagramma dei casi d'uso
- 2.2.2 Diagramma delle classi
 - 2.2.2.1 Descrizione delle interfacce
 - 2.2.2.2 Descrizione delle strutture dati
 - 2.2.2.3 Descrizione delle classi

2.3 Prototipi Interfaccia Grafica

3 Seconda Applicazione Zone

3.1 Progettazione Architetturale

- 3.1.1 Organizzazione del Sistema
- 3.1.2 Scomposizione Modulare
 - 3.1.2.1 Moduli dei Sottosistemi

3.2 Progettazione Orientata agli Oggetti

- 3.2.1 Diagramma dei casi d'uso
- 3.2.2 Diagramma delle classi
 - 3.2.2.1 Descrizione delle interfacce
 - 3.2.2.2 Descrizione delle strutture dati
 - 3.2.2.3 Descrizione degli enumeratori
 - 3.2.2.4 Descrizione delle classi

3.3 Prototipi Interfaccia Grafica

Prefazione

Lettura del documento

Il seguente documento di progettazione fa riferimento ai due documenti: Piano di Progetto 1.0, Documento di Analisi e Specifica dei Requisiti 1.1 che presenta una versione rivisitata e semplificata dei diagrammi dei casi d'uso.

Versione

Questa prima versione del documento presenta delle incertezze nella modellazione architettonale orientata agli oggetti, e ampie lacune evidenti nella fase di progettazione del sistema della seconda applicazione per smartphone (Ratondo).

1 Introduzione

1.1 Scopo del documento di progettazione

Lo scopo del seguente documento di progettazione del software consiste nel dare un'organizzazione logica al software, facendo uso di modelli UML, che sia integrabile con la specifica dei requisiti, cioè soddisfi i requisiti di sistema funzionali e non funzionali, e di riferimento e facile utilizzo nel processo di sviluppo del software. Per entrambe le applicazioni bisognerà stabilire un'organizzazione che soddisfi i requisiti di sistema.

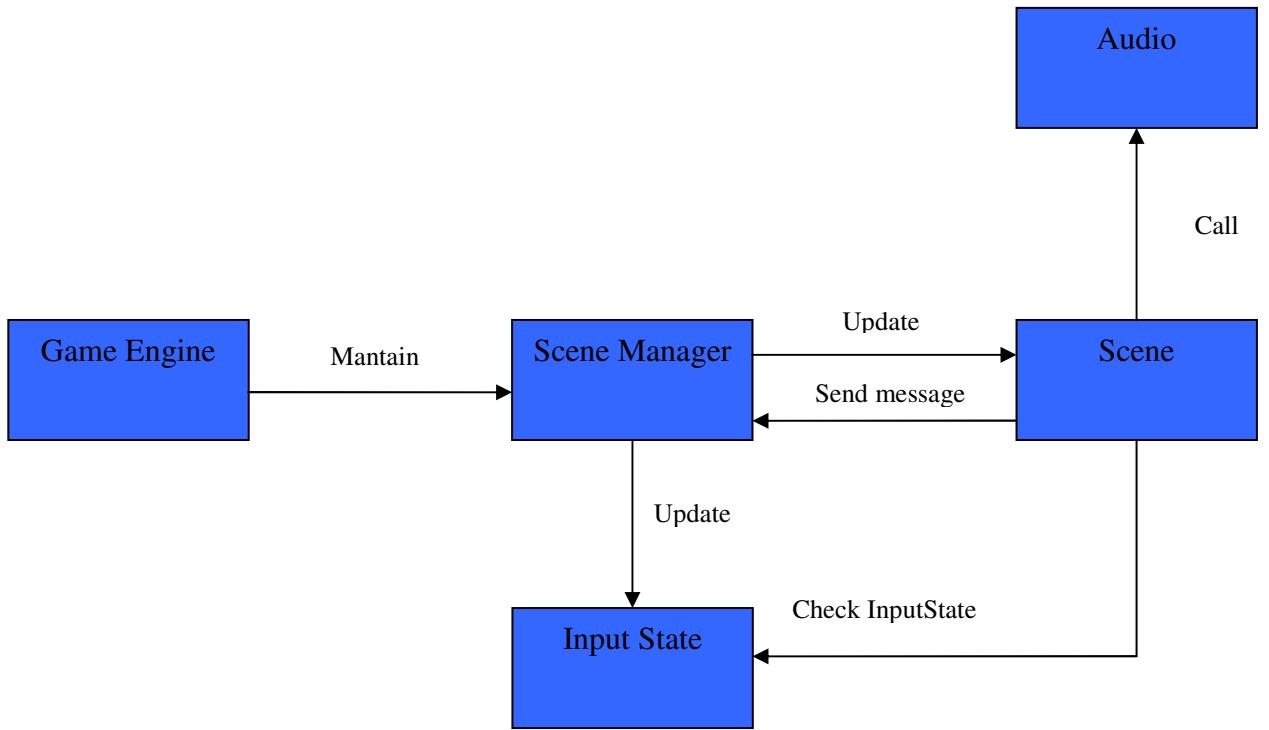
Nel dominio di applicazione del sistema si individuano sistemi con architetture simili che ne riflettono i concetti fondamentali, riutilizzando le classi di applicazione più generali in comune con il sistema. Si dovrà anche decidere per l'architettura di sistema delle due applicazioni il modello architettonale da adottare o schema dell'organizzazione del sistema. Dopo aver scelto l'organizzazione generale del sistema si deve decidere l'approccio da usare per la scomposizione dei sottosistemi in moduli.

2 Prima Applicazione Quiz Time

2.1 Progettazione Architettonale

2.1.1 Organizzazione del Sistema

Figura 3.1 Architettura del sistema dell'applicazione Zone



Sottosistemi

Breve descrizione dei sottosistemi di cui è composto il sistema dell'applicazione Zone in Figura 3.1.

GameEngine: Fornisce l'inizializzazione, la logica di gioco e il codice di rendering di base per un dispositivo grafico. Si occupa della creazione e mantenimento di uno Scene Manager.

SceneManager: Gestisce il ciclo di vita delle scene, e dell'avanzamento del loro stato.

Scene: Crea e gestisce gli elementi di una scena su schermo.

InputState: Legge l'input, tenendo traccia dello stato corrente e precedente del dispositivo di input.

Audio: Gestisce tutti i suoni.

Settings: Mantiene i dati delle impostazioni e carica i dati di configurazione.

2.1.2 Scomposizione Modulare

Scomponiamo i sottosistemi in moduli usando una scomposizione orientata agli oggetti.

Con questo approccio i moduli dei sottosistemi verranno rappresentati come oggetti, che richiedono servizi offerti da altri oggetti.

Usiamo i diagrammi dei casi d'uso in UML per mostrare le classi di sistema.

2.1.2.1 Moduli dei Sottosistemi

Insieme dei moduli o delle classi di oggetti individuate nei sottosistemi.

Sottosistema GameEngine

Classi:

- Microsoft.Xna.Framework.Game
- Zone
 - GraphicDeviceManager

Sottosistema SceneManager

Classi:

- ScreenManager

Sottosistema InputState:

Classi:

- InputState

Sottosistema Settings:

Classi:

- SettingsManager

Sottosistema Audio:

Classi:

- AudioManager

Sottosistema Scene:

Classi:

- GameScreen
 - BasicMenu
 - MainMenuScreen
 - PauseScreen
 - BackgroundScreen
 - GameplayScreen
 - Level
 - LoadingScreen
 - MenuScreen
 - HighScoreScreen

2.2 Progettazione orientata agli oggetti

2.2.1 Diagramma dei casi d'uso

Diagramma Casi d'uso Menu Principale

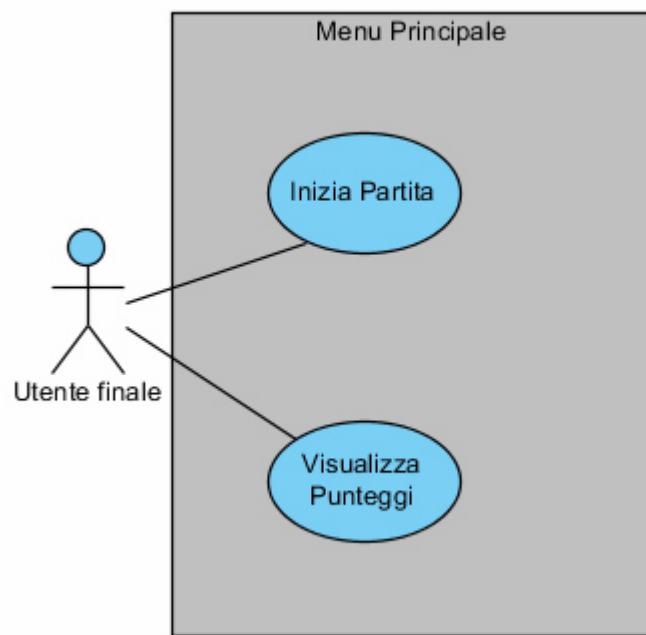
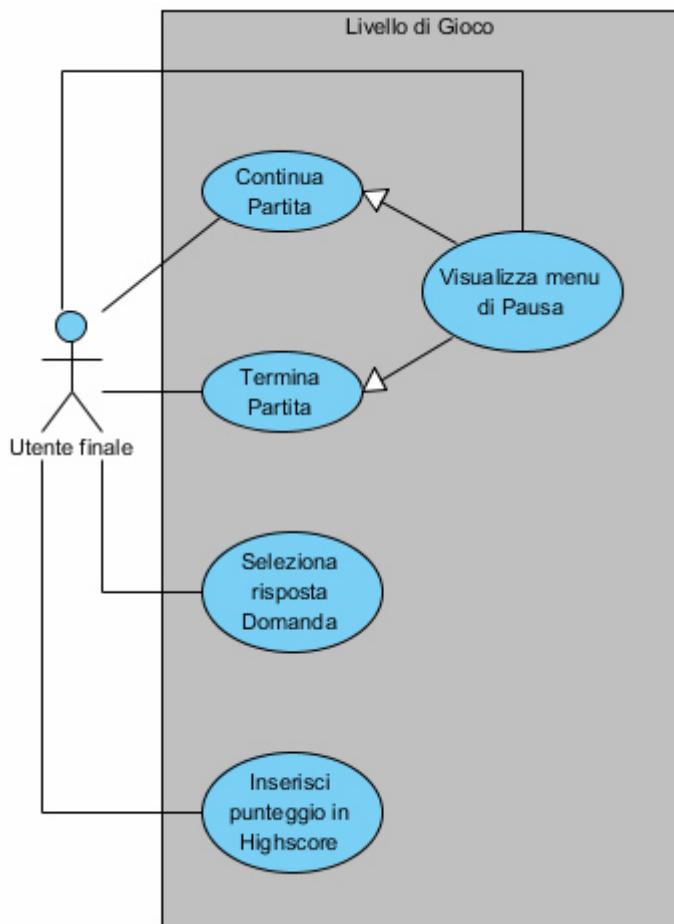
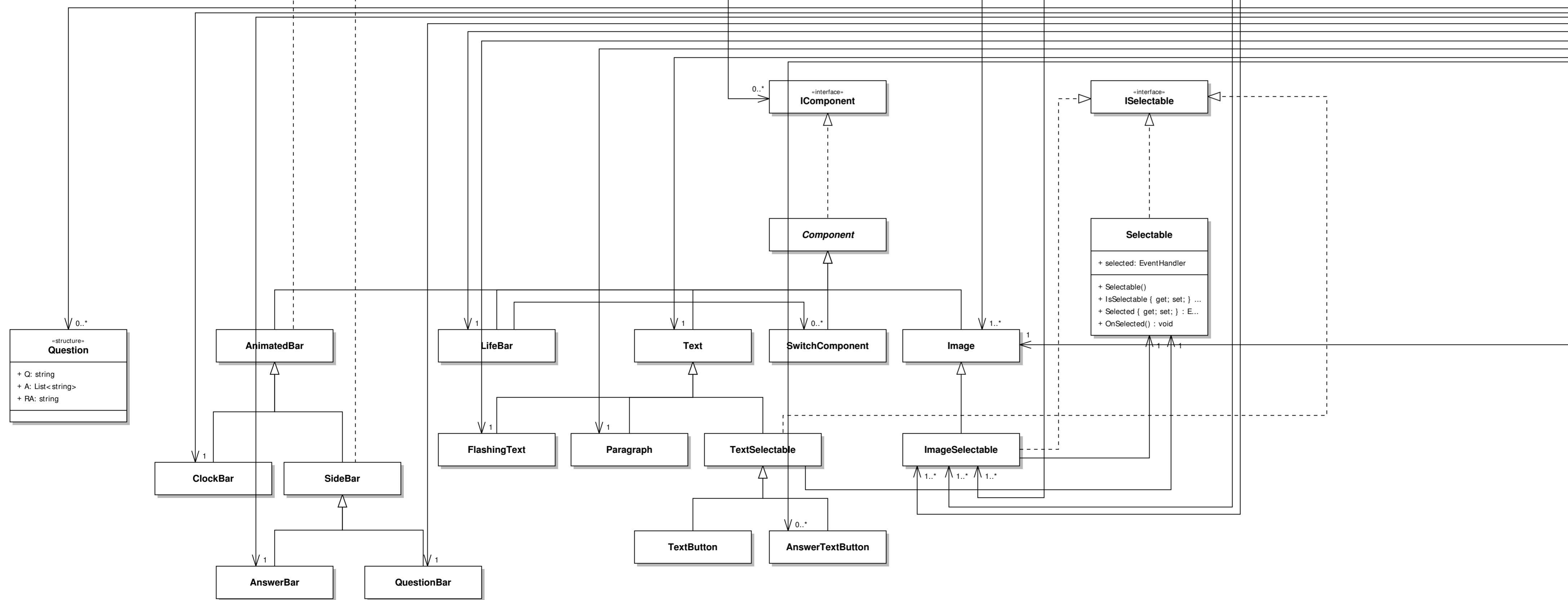
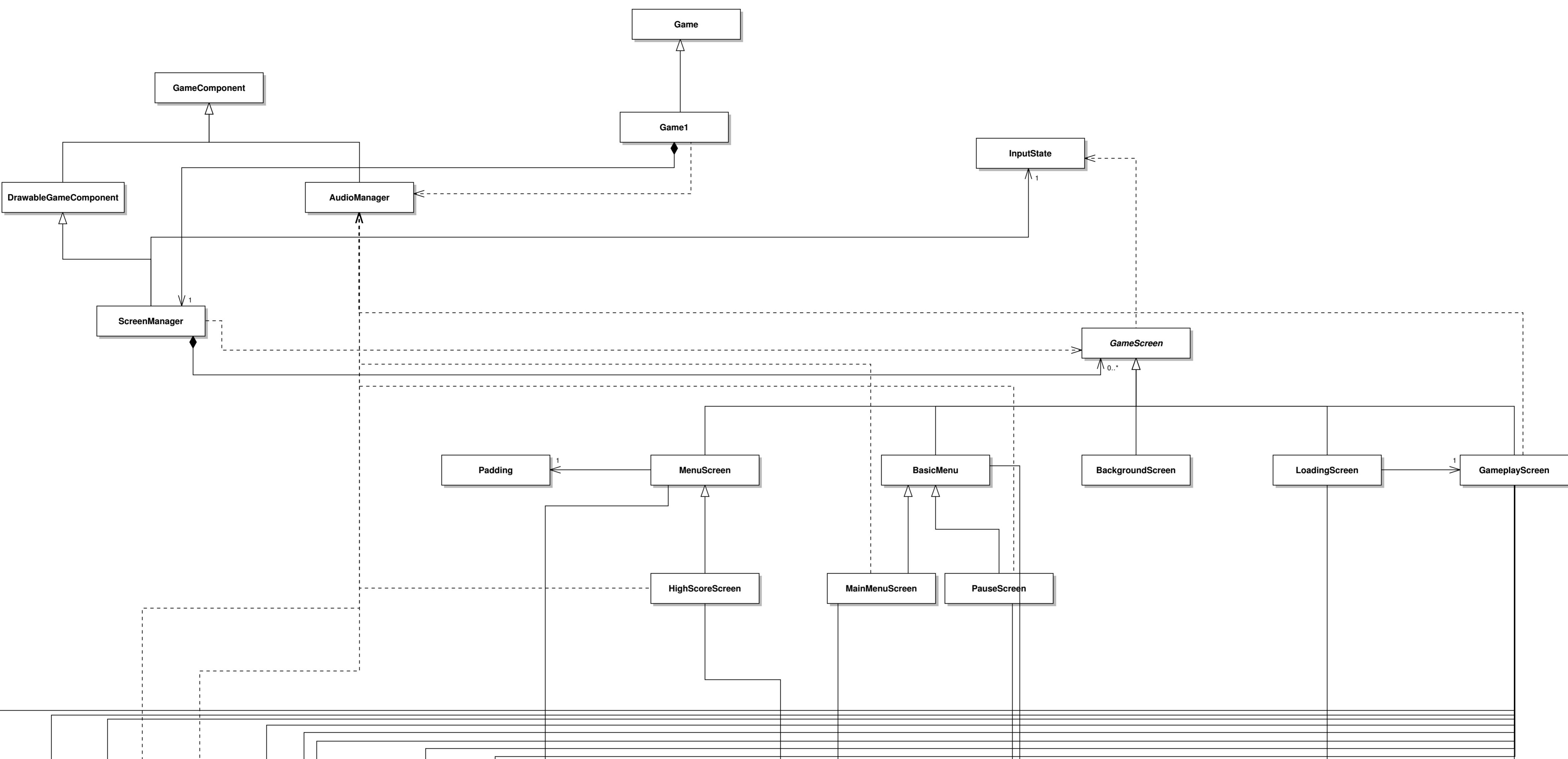


Diagramma casi d'uso Livello di gioco



2.2.2 Diagramma delle classi



2.2.2.1 Descrizione delle interfacce

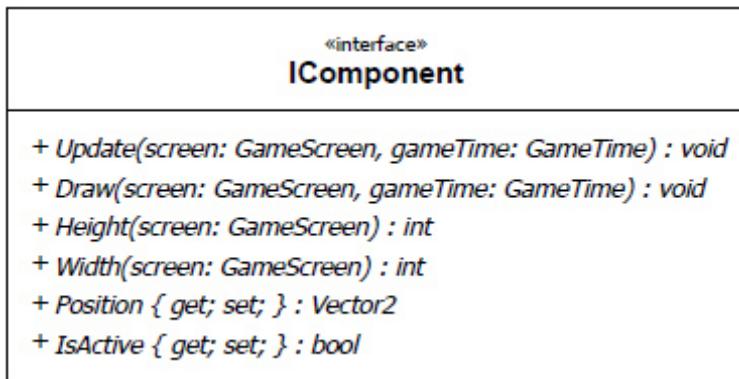
IComponent :

Namespace: QuizTime

Inherit: /

Descrizione:

Interfaccia implementata da tutte le classi per la realizzazione dei componenti con una dimensione(altezza, larghezza) che devono essere disegnati su schermo, e il cui stato necessita in vari casi di essere aggiornato in risposta al sistema.



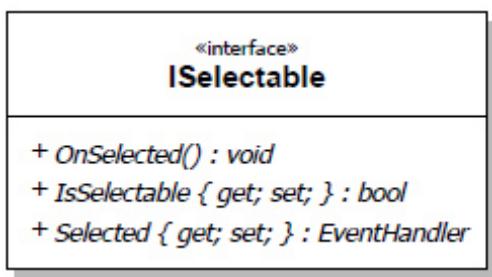
ISelectable :

Namespace: QuizTime

Inherit: /

Descrizione:

Interfaccia implementata dalle classi che devono gestire gli input dati dall'utente.



2.2.2.2 Descrizione delle strutture dati

Question:

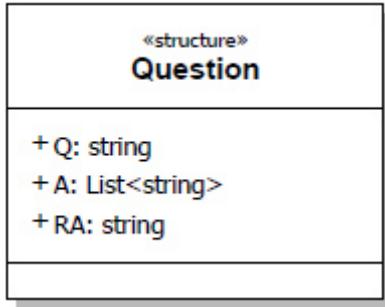
Struttura utilizzata per salvare le informazioni di una “query”.

Campi della struttura:

string Q → Memorizza il testo della domanda.

List<string> A → Memorizza il testo delle risposte.

string RA → Memorizza il testo della risposta corretta.



2.2.2.3 Descrizione delle classi

Game

Namespace: Microsoft.Xna.Framework

Inherit: /

Implement: IDisposable

Descrizione:

Classe contenuta nella libreria XNA Framework, fornisce l'inizializzazione, la logica di gioco e il codice di rendering di base per un dispositivo grafico.

GameComponent

Namespace: Microsoft.Xna.Framework

Inherit: IGameComponent

Implement: IUpdateable, IDisposable

Descrizione:

Classe base per tutti i componenti di gioco di XNA Framework.

Un componente di gioco che estende questa classe dispone di metodi di aggiornamento e inizializzazione chiamati dai metodi Game.Initialize, Game.Update qualora sia stato registrato con in metodo Game.Component.Add.

DrawableGameComponent

Namespace: Microsoft.Xna.Framework

Inherit: GameComponent

Implement: IDrawable

Descrizione:

Componente del gioco a cui viene inviata una notifica quando deve disegnarsi automaticamente.

Game1

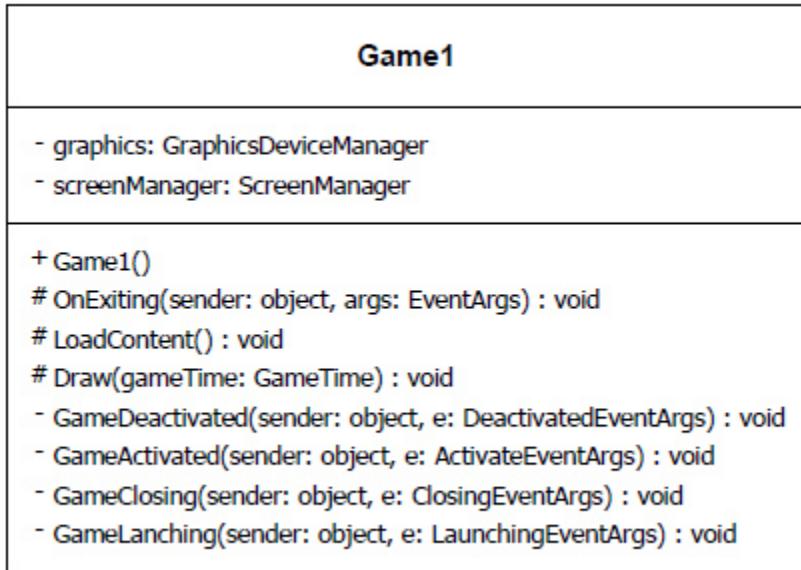
Namespace: QuizTime

Inherit: Game

Implement: /

Descrizione:

Contiene un'istanza dell'oggetto GraphicDeviceManager che consente di configurare e gestire il dispositivo grafico, e un'istanza dell'oggetto ScreenManager usato per gestire le scene.



ScreenManager

Namespace: QuizTime

Inherit: DrawableGameComponent

Implement: /

Descrizione:

Contiene un elenco di oggetti di tipo GameScreen, di ogni GameScreen nella lista viene chiamato il metodo di Update a cui viene passato la variabile booleana coveredByOtherScreen che serve a stabilire se lo screen debba essere nascosto e lasciare posto ad un altro screen. Inoltre richiama il metodo HandleInput dello screen attivo per la gestione dell'input, se non ci sono altri screen che hanno il focus.

ScreenManager	
- screens: List<GameScreen>	
- screensToUpdate: List<GameScreen>	
- input: InputState	
- spriteBatch: SpriteBatch	
- font: SpriteFont	
- blankTexture: Texture2D	
- siInitialized: bool	
+ SpriteBatch { get; } : SpriteBatch	
+ Font { get; } : SpriteFont	
+ BlankTexture { get; } : Texture2D	
+ ScreenManager(game: Game)	
+ Initialize() : void	
# LoadContent() : void	
# UnloadContent() : void	
+ Update(gameTime: GameTime) : void	
+ Draw(gaemTime: GameTime) : void	
+ AddScreen(screen: GameScreen) : void	
+ RemoveScreen(screen: GameScreen) : void	
+ GetScreens() : GameScreen[]	
+ FadeBackBufferToBlack(alpha: float) : void	
+ SerializeState() : void	
+ DeserializeState() : bool	
- DeleteState(storage: IsolatedStorageFile) : void	

InputState

Namespace: QuizTime

Inherit: /

Implement: /

Descrizione:

Questa classe legge gli input tenendo traccia del corrente e precedente stato dell'input, è utile per rilevare gli input dello smartphone, ma può essere esteso anche a dispositivi come tastiera, mouse e controller.

InputState

```
+ MaxInputs: int = 4
+ CurrentGamePadStates: GamePadState[]
+ LastGamePadStates: KeyboardState[]
+ GamePadWasConnected: bool[]
+ TouchState: TouchCollection
+ Gestures: List<GestureSample> = new List<GestureSample>()

+ InputState()
+ Update() : void
# CheckButton(button: Button, controllingPlayer: PlayerIndex?) : bool
+ isNewButtonPress(button: Buttons, controllingPlayer: PlayerIndex?, out playerIndex: PlayerIndex) : bool
+ IsMenuSelect() : bool
+ IsMenuCancel() : bool
+ isPauseGame() : bool
```

AudioManager

Namespace: QuizTime

Inherit: GameComponent

Implement: /

Descrizione:

Carica tutti i suoni e le musiche, e implementa i metodi per la loro gestione.
Per questa classe è stato usato il pattern Singleton.

AudioManager

```
- isActive: bool  
- isInitialized: bool  
- audioManager: AudioManager = null  
- soundAssetLocation: string = "Sounds/"  
- soundBank: Dictionary<string, SoundEffectInstance>  
- musicBank: Dictionary<string, Song>
```

```
- AudioManager(game: Game)  
+ Instance { get; } : AudioManager  
+ IsInitialized { get; } : bool  
+ IsActive { get; set; } : bool  
+ Initialize(game: Game) : void  
+ LoadSound(contentName: string, alias: string) : void  
+ LoadSong(contentName: string, alias: string) : void  
+ LoadSounds() : void  
+ LoadMusic() : void  
+ this[string soundName] { get; } : SoundEffectInstance  
+ PlaySound(soundName: string) : void  
+ PlaySound(soundName: string, isLooped: bool) : void  
+ PlaySound(soundName: string, isLooped: bool, volume: float) : void  
+ StopSound(soundName: string) : void  
+ StopSounds() : void  
+ PauseResumeSounds(resumeSounds: bool) : void  
+ PlayMusic(musicSoundName: string) : void  
+ StopMusic() : void  
+ Enable(state: bool) : void  
# Dispose(disposing: bool) : void
```

GameScreen

Namespace: QuizTime

Inherit: /

Implement: /

Descrizione:

E' la classe astratta che rappresenta una scena su schermo.

GameScreen

```
# contentDynamic: ContentManager
# LanguageDefinitions: Dictionary<string,string>
- isPopup: bool = false
- screenSize: ScreenState = ScreenState.TransitionOn
- isExiting: bool = false
- otherScreenHasFocus: bool
- screenManager: ScreenManager
- enabledGesture: GestureType = GestureType.None
- isSerializable: bool = true

+ isPopup { get; set; } : bool
+ ScreenState { get; set; } : ScreenState
+ isExiting { get; set; } : bool
+ isActive { get; } : bool
+ ScreenManager { get; set; } : ScreenManager
+ EnabledGestures { get; set; } : GestureType
+ isSerializable { get; set; } : bool
+ Initialize() : void
+ LoadContent() : void
+ UnloadContent() : void
+ Update(gameTime: GameTime, otherScreenHasFocus: bool, coveredByOtherScreen: bool) : void
+ Draw(gameTime: GameTime) : void
+ HandleInput(input: InputState) : void
+ Serialize(stream: Stream) : void
+ Deserialize(stream: Stream) : void
+ ExitScreen() : void
+ LoadAssets() : void
+ Load<T>(assetName: string) : T
# ReplaceForwardScreens(screens: List<GameScreen>) : void
# ReplaceAllScreens(screens: List<GameScreen>) : void
```

Classi Ereditarie di GameScene

La classe GameScreen è ereditata direttamente dalle classi:

- MenuScreen
- BasicMenu
- BackgroundScreen
- LoadingScreen
- GameplayScreen

BackgroundScreen:

Namespace: QuizTime

Inherit: GameScreen

Implement: /

Descrizione:

Rappresenta il background sullo schermo, non può gestire gli input e consente agli altri screen di sovrapporsi e gestire gli input

BackgroundScreen

- backgroundTexture: Texture2D
- backgroundName: string

+ BackgroundScreen(backgroundName: string)
+ LoadContent() : void
+ Update(gameTime: GameTime, otherScreenHasFocus: bool, coveredByOtherScreen: bool) : void
+ Draw(gameTime: GameTime) : void

GameplayScreen:

Namespace: QuizTime

Inherit: GameScreen

Implement: /

Descrizione:

E' la classe dove viene eseguita la logica del gioco

GameplayScreen	
<ul style="list-style-type: none"> - IsActive: bool - random: Random - queryList: List<List<Question>> - outerListIndex: int - innerListIndex: int - currentRightAnswer: string - maxQuestionPerList: int - isShowComponents: bool - userTapToStar: bool - isUserWon: bool - gameEnded: bool - userTapToExit: bool - hasAnswered: bool - insertInHighScore: bool - nextQuery: bool - isLifeLost: bool - isPlaySound: bool - isAlarmClock: bool - moveToHighScore: bool - rightAnswers: int - currentQuestionNumber: int - maxQuestionNumber: int - questionBar: QuestionBar - answerBar: AnswerBar - lifeBar: LifeBar - clockBar: ClockBar - flashingText: FlashingText - touchNotice: Text - questionParagraph: Paragraph - answerButtons: List<AnswerTextButton> - timeToWait: TimeSpan - timeElapsed: TimeSpan - flashingDuration: TimeSpan - maxClockTime: TimeSpan - isResuming: bool 	
<ul style="list-style-type: none"> + GameplayScreen() + IsActive { get; set; } : bool + RightAnswers { get; } : int - IsOpenComponentsBar { get; } : bool - IsCloseComponentsBar { get; } : bool - ComponentsBarIsOpened { get; } : bool - ComponentsBarIsClosed { get; } : bool - IsOpenedComponentsGrill { get; } : bool - ComponentsGrillIsOpened { get; } : bool - ComponentsGrillIsClosed { get; } : bool + LoadContent() : void + LoadAssets() : void + LoadAssets() : void - LoadQuestionsFromXML() : void - LoadTextures() : void - CreateGameComponents() : void + InitializeQueryComponents() : void + InitializeAnimationFields() : void + HandleInput(input: InputState) : void + Update(gameTime: GameTime, otherScreenHasFocus: bool, coveredByOtherScreen: bool) : void + Draw(gameTime: GameTime) : void # GetEntryHitBounds(component: IComponent) : Rectangle - CheckIfCurrentGameFinished() : bool - CheckIsInHighScore() : bool - DrawGameEndIfNecessary() : void - PauseCurrentGame() : void 	

LoadingScreen:

Namespace: QuizTime

Inherit: GameScreen

Implement: /

Descrizione:

Indica la schermata di caricamento, viene lanciata prima del GameplayScreen per consentire il caricamento dei contenuti di gioco (Texture2D, SpriteFont) e l'inizializzazione delle variabili.

LoadingScreen	
- circleTexture: Texture2D	
- circle: Image	
- speed: float	
- isLoading: bool	
- gameplayScreen: GameplayScreen	
- loadingThread: Thread	
+ LoadContent() : void	
+ Update(gameTime: GameTime, otherScreenHasFocus: bool, coveredByOtherScreen: bool) : void	
+ Draw(gameTime: GameTime) : void	

MenuScreen:

Namespace: QuizTime

Inherit: GameScreen

Implement: /

Descrizione:

Classe astratta che estende Gamescreen. Possiede un elenco di oggetti di tipo IComponent, di cui gestisce gli eventi.

MenuScreen
<ul style="list-style-type: none"> - components: List<IComponent> # padding: Padding
<ul style="list-style-type: none"> + MenuScreen() + LoadContent() : void + HandleInput(input: InputState) : void + Update(gameTime: GameTime, otherScreenHasFocus: bool, coveredByOtherScreen: bool) : void + Draw(gameTime: GameTime) : void # UpdateComponentsLocation() : void # OnSelectEntry(entryIndex: int) : void # OnCancel() : void # AddComponent(component: IComponent) : void

BasicMenu:

Namespace: QuizTime

Inherit: GameScreen

Implement: /

Descrizione:

Classe che costruisce le linee guida per la realizzazione delle schermate del menu principale. Ereditata dalle classi MainMenuScreen e PauseScreen per la realizzazione della schermata del main menu e del menu di pausa.

BasicMenu

```
# buttonDistance: int
# sideBarAnimationStep: int
# blankTexture: Texture2D
# buttonMenuTexture: Texture2D
# titleMenuTexture: Texture2D
# menuFont: SpriteFont
# textColor: Color
# alphaChannel: float
# title: Image
- animateSideBar: bool
- sideBarIsActive: bool
- sideBarInTransition: bool
- sideBarHitFinalPosition: bool
- sideBarList: List<IComponent>
- SideBarsClosedPosition: List<Vector2>
- SideBarOpenedPosition: List<Vector2>

+ BasicMenu()
# Title { get; } : IComponent
# MenuFont { get; set; } : SpriteFont
+ LoadContent() : void
+ HandleInput(input: InputState) : void
+ Update(gameTime: GameTime, otherScreenHasFocus: bool, coveredByOtherScreen: bool) : void
+ Draw(gameTime: GameTime) : void
- AnimateSideBar() : void
# AddSideBarComponent(component: IComponent) : void
# GetEntryHitBounds(component: IComponent) : Rectangle
# OnCancel() : void
```

Classi Ereditarie di MenuScreen

La classe MenuScreen è estesa dalle seguenti classi:

- HighScoreMenuScreen

HighScoreScreen:

Namespace: QuizTime

Inherit: MenuScreen

Implement: /

Descrizione:

classe per la gestione della schermata dei punteggi.

HighScoreScreen

```
- leftDoorTexture: Texture2D
- rightDoorTexture: Texture2D
- leftDoor: Image
- rightDoor: Image
- doorAnimationStep: int
- animateDoors: bool
- doorsInTransition: bool
- doorsHitFinalPosition: bool
- leftDoorOpenedPosition: Vector2
- leftDoorClosedPosition: Vector2
- rightDoorOpenedPosition: Vector2
- rightDoorClosedPosition: Vector2
- HighScoreDataDestination: string = "HighscoreData.sav"
- blankTexture: Texture2D
- menuFont: SpriteFont
- scoreFont: SpriteFont
- highscorePlaces: int = 6
+ highScore: List<KeyValuePair<string, int>>
- drawHighScore: bool
- numberPlaceMapping: Dictionary<int, string>

+ HighScoreScreen()
+ LoadContent() : void
+ HandleInput(input: InputState) : void
+ Update(gameTime: GameTime, otherScreenHasFocus: bool, coveredByOtherScreen: bool) : void
+ Draw(gameTime: GameTime) : void
- Exit() : void
- AnimateDoors() : void
+ IsInHighscores(score: int) : bool
+ PutHighScore(playerName: string, score: int) : void
- OrderGameScore() : void
+ SaveHighscore() : void
+ LoadHighscores() : void
- GetPlaceString(number: int) : string
- initializeMapping() : void
- InitializeScore() : List<KeyValuePair<string, int>>
+ NewMethod() : void
```

Classi Ereditarie di BasicMenu

La classe BasicMenu è estesa dalle seguenti classi:

- MainMenuItem
- PauseScreen

MainMenuScreen:

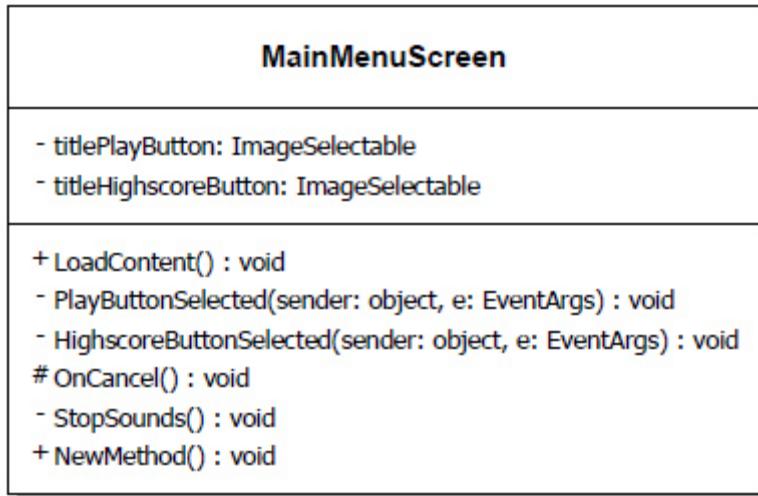
Namespace: QuizTime

Inherit: BasicMenu

Implement: /

Descrizione:

Classe per la realizzazione della schermata del menu principale (main menu).



PauseScreen:

Namespace: QuizTime

Inherit: BasicMenu

Implement: /

Descrizione:

Classe per la realizzazione del menu di pausa del gioco.

PauseScreen
<ul style="list-style-type: none"> - titleContinueButton: ImageSelectable - titleQuitButton: ImageSelectable - isResuming: bool - checkHighscore: bool - moveToHighScore: bool - moveToMainMenu: bool
<ul style="list-style-type: none"> + PauseScreen(isResuming: bool) + LoadContent() : void + Update(gameTime: GameTime, otherScreenHasFocus: bool, coveredByOtherScreen: bool) : void - ContinueButtonSelected(sender: object, e: EventArgs) : void - QuitButtonSelected(sender: object, e: EventArgs) : void # OnCancel() : void - GetGameplayScreen() : GameplayScreen

Altre Classi

Component:

Namespace: QuizTime

Inherit: /

Implement: IComponent

Descrizione:

Classe di base estesa da tutte le classi che devono rappresentare un oggetto o insieme di oggetti su schermo.

Component
<pre># index: int # position: Vector2 # sourceRectangle: Rectangle? # color: Color # origin: Vector2 # scale: float # rotation: float # alphaChannel: float # effects: SpriteEffects</pre>
<pre>+ Component(index: int) + IsActive { get; set; } : bool + Index { set; } : int + SourceRectangle { get; set; } : Rectangle? + Color { get; set; } : Color + Origin { get; set; } : Vector2 + Scale { get; set; } : float + Rotation { get; set; } : float + AlphaChannel { get; set; } : float + Effects { get; set; } : SpriteEffects + Position { get; set; } : Vector2 + Update(screen: GameScreen, gameTime: GameTime) : void + Draw(screen: GameScreen, gameTime: GameTime) : void + Height(screen: GameScreen) : int + Width(screen: GameScreen) : int</pre>

La classe Component è estesa direttamente dalle seguenti classi:

- AnimatedBar
- LifeBar
- Text
- SwitchComponent
- Image

Selectable:

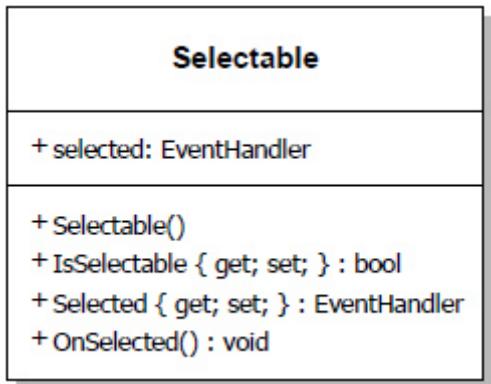
Namespace: QuizTime

Inherit: /

Implement: ISelectable

Descrizione:

Classe per la gestione di eventi.



Classi ereditarie di Component

Classi ereditarie dirette di Component

AnimatedBar:

Namespace: QuizTime

Inherit: Component

Implement: /

Descrizione:

Classe base per la rappresentazione di barre animate, cioè che viaggiano lungo un percorso fisso, da un punto iniziale fisso a un punto finale fisso e viceversa.

AnimatedBar

```
# position: Vector2
- closedPosition: Vector2
- openedPosition: Vector2
- barIsOpened: bool
- isAnimateBar: bool
- isOpenBar: bool
- barHitFinalPosition: bool
- barAnimationStep: int
- newField: int
```

```
+ AnimatedBar(closedPosition: Vector2, openedPosition: Vector2, barIsOpened: bool)
+ AnimatedBar(index: int, closedPosition: Vector2, openedPosition: Vector2, barIsOpened: bool)
# Position { get; set; } : Vector2
+ BarIsOpened { get; set; } : bool
+ IsOpenBar { get; set; } : bool
+ BarAnimationStep { get; set; } : int
+ Update(screen: GameScreen, gameTime: GameTime) : void
+ OpenBar(open: bool) : void
- AnimateBar(screen: GameScreen) : void
```

La classe AnimatedBar è estesa direttamente dalle seguenti classi:

- ClockBar
- SideBar

LifeBar:

Namespace: QuizTime

Inherit: Component

Implement: /

Descrizione:

Gestisce il numero di errori commessi dall'utente in gioco, aggiornando gli elementi grafici che fungono da elemento visivo per informare l'utente degli errori commessi.

LifeBar

```
- switchComponents: SwitchComponent[]
- imageOn: Texture2D
- imageOff: Texture2D
- maxLife: int
- numLife: int
- switchOn: bool

+ LifeBar(maxLife: int, imageOn: Texture2D, imageOff: Texture2D, switchOn: bool)
+ NumLife { get; } : int
+ Update(screen: GameScreen, gameTime: GameTime) : void
+ Draw(screen: GameScreen, gameTime: GameTime) : void
+ Height(screen: GameScreen) : int
+ Width(screen: GameScreen, gameTime: GameTime) : int
+ IsLifeEmpty() : bool
+ RemoveLife() : void
```

Text:

Namespace: QuizTime

Inherit: Component

Implement: /

Descrizione:

Classe per la rappresentazione di elementi testuali.

Text

```
- textContents: string
- font: SpriteFont

+ Text(textContents: string, font: SpriteFont)
+ Text(index: int, textContents: string, font: SpriteFont)
+ TextContents { get; set; } : string
+ Font { get; set; } : SpriteFont
+ Update(screen: GameScreen, gameTime: GameTime) : void
+ Draw(screen: GameScreen, gameTime: GameTime) : void
+ Height(screen: GameScreen) : int
+ Width(screen: GameScreen) : int
```

La classe Text è estesa direttamente dalle seguenti classi:

- Flashing
- Paragraph
- TextSelectable

SwitchComponent:

Namespace: QuizTime

Inherit: Component

Implement: /

Descrizione:

Classe per la rappresentazione di elementi a due stati.

SwitchComponent	
-	switchOn: bool
-	imageOn: Texture2D
-	imageOff: Texture2D
+	SwitchComponent(imageOn: Texture2D, imageOff: Texture2D, switchOn: bool)
+	SwitchComponent(index: int, imageOn: Texture2D, imageOff: Texture2D, switchOn: bool)
+	SwitchOn { get; set; } : bool
+	Update(screen: GameScreen, gameTime: GameTime) : void
+	Draw(screen: GameScreen, gameTime: GameTime) : void
+	Height(screen: GameScreen) : int
+	Width(screen: GameScreen) : int

Image:

Namespace: QuizTime

Inherit: Component

Implement: /

Descrizione:

Classe per la rappresentazione di elementi immagine.

Image
- <code>imageContents: Texture2D</code>
+ <code>Image(imageContents: Texture2D)</code>
+ <code>Image(index: int, imageContents: Texture2D)</code>
+ <code>ImageContents { get; set; } : Texture2D</code>
+ <code>Bounds { get; } : Rectangle</code>
+ <code>Update(screen: GameScreen, gameTime: GameTime) : void</code>
+ <code>Draw(screen: GameScreen, gameTime: GameTime) : void</code>
+ <code>Height(screen: GameScreen) : int</code>
+ <code>Width(screen: GameScreen) : int</code>

La classe Image è estesa direttamente dalle seguenti classi:

- `ImageSelectable`

Classi ereditarie dirette di AnimatedBar

ClockBar:

Namespace: QuizTime

Inherit: `AnimatedBar`

Implement: /

Descrizione:

Classe per la realizzazione della barra per il controllo del tempo in gioco.

ClockBar
- <code>clockBarTexture: Texture2D</code>
- <code>maxClockTime: TimeSpan</code>
- <code>clockTime: TimeSpan</code>
- <code>font: SpriteFont</code>
- <code>isTimeOut: bool</code>
- <code>isOutOfTime: bool</code>
+ <code>ClockBar(font: SpriteFont, clockBarTexture: Texture2D, maxClockTime: TimeSpan, isTimeOut: bool, closedPosition: Vector2, openedPosition: Vector2, barIsOpened: bool)</code>
+ <code>ClockBar(index: int, font: SpriteFont, clockBarTexture: Texture2D, maxClockTime: TimeSpan, isTimeOut: bool, closedPosition: Vector2, openedPosition: Vector2, barIsOpened: bool)</code>
+ <code>ClockBarTexture { get; set; } : Texture2D</code>
+ <code>MaxClockTime { get; set; } : TimeSpan</code>
+ <code>ClockTime { get; set; } : TimeSpan</code>
+ <code>IsTimeOut { get; set; } : bool</code>
+ <code>IsOutOfTime { get; } : bool</code>
+ <code>Update(screen: GameScreen, gameTime: GameTime) : void</code>
+ <code>Draw(screen: GameScreen, gameTime: GameTime) : void</code>
+ <code>Height(screen: GameScreen) : int</code>
+ <code>Width(screen: GameScreen) : int</code>
+ <code>ResetClock() : void</code>
- <code>UpdateClockText() : String</code>
- <code>UpdateClockTextPosition() : Vector2</code>

SideBar:

Namespace: QuizTime

Inherit: AnimatedBar

Implement: /

Descrizione:

Classe base per la realizzazione di una barra animata.

SideBar

```
- outerBlockTexture: Texture2D
- innerBlockTexture: Texture2D
- grillBlockTexture: Texture2D
# outerBlock: Image
# innerBlock: Image
# grillBlock: Image
- grillBlockClosedDimension: Rectangle
- grillBlockOpenedDimension: Rectangle
- grillIsOpened: bool
- isAnimateGrill: bool
- isOpenGrill: bool
- grillHitFinalPosition: bool
+ grillAnimationStep: int

+ SideBar(outerBlockTexture: Texture2D, innerBlockTexture: Texture2D, closedPosition: Vector2, openedPosition: Vector2, sideBarIsOpen: bool)
+ SideBar(index: int, outerBlockTexture: Texture2D, innerBlockTexture: Texture2D, closedPosition: Vector2, openedPosition: Vector2, sideBarIsOpen: bool)
+ OuterSpace { get; set; } : Vector2
+ innerSpace { get; set; } : Vector2
+ grillSpace { get; set; } : Vector2
+ GrillIsOpened { get; set; } : bool
+ IsOpenGrill { get; set; } : bool
- GrillAnimationStep { get; set; } : int
- InitializeConstructor(outerBlockTexture: Texture2D, grillBlockTexture: Texture2D, grillIsOpened: bool) : void
+ Update(screen: GameScreen, gameTime: GameTime) : void
+ Draw(screen: GameScreen, gameTime: GameTime) : void
+ Height(screen: GameScreen) : int
+ Width(screen: GameScreen) : int
# UpdateDependingPositionComponent(screen: GameScreen, gameTime: GameTime) : void
# DrawContent(screen: GameScreen, gameTime: GameTime) : void
- AnimateGrill(screen: GameScreen) : void
```

Classi ereditarie dirette di SideBar

AnswerBar:

Namespace: QuizTime

Inherit: SideBar

Implement: /

Descrizione:

Classe per la realizzazione del riquadro animato che contiene il testo delle risposte.

AnswerBar
<pre> - answerButtons: List<AnswerTextButton> + AnswerBar(outerBlockTexture: Texture2D, innerBlockTexture: Texture2D, grillBlockTexture: Texture2D, closedPosition: Vector2, openedPosition: Vector2, sideBarIsOpened: bool, grillIsOpened: bool) + AnswerBar(index: int, outerBlockTexture: Texture2D, innerBlockTexture: Texture2D, grillBlockTexture: Texture2D, closedPosition: Vector2, openedPosition: Vector2, sideBarIsOpened: bool, grillIsOpened: bool) + AnswerButtons { get; set; } : List<AnswerTextButton> + answerButtonSpace { get; set; } : Vector2 # UpdateDependingPositionComponent(screen: GameScreen, gameTime: GameTime) : void # DrawContent(screen: GameScreen, gameTime: GameTime) : void </pre>

QuestionBar:

Namespace: QuizTime

Inherit: SideBar

Implement: /

Descrizione:

Classe per la realizzazione del riquadro animato che contiene il paragrafo contenente il testo della domanda da visualizzare.

QuestionBar
<pre> - questionParagraph: Paragraph + QuestionBar(outerBlockTexture: Texture2D, innerBlockTexture: Texture2D, grillBlockTexture: Texture2D, closedPosition: Vector2, openedPosition: Vector2, sideBarIsOpened: bool, grillIsOpened: bool) + QuestionBar(index: int, outerBlockTexture: Texture2D, innerBlockTexture: Texture2D, grillBlockTexture: Texture2D, closedPosition: Vector2, openedPosition: Vector2, sideBarIsOpened: bool, grillIsOpened: bool) + QuestionParagraph { get; set; } : Rectangle + paragraphSpace { get; set; } : Vector2 # UpdateDependingPositionComponent(screen: GameScreen, gameTime: GameTime) : void # DrawContent(screen: GameScreen, gameTime: GameTime) : void </pre>

Classi ereditarie dirette di Text

FlashingText:

Namespace: QuizTime

Inherit: Text

Implement: /

Descrizione:

Classe per la realizzazione di un testo lampeggiante.

FlashingText

```
- flashTime: TimeSpan  
- flashingDuration: TimeSpan  
  
+ FlashingText(text: string, font: SpriteFont)  
+ FlashingText(index: int, text: string, font: SpriteFont)  
+ FlashingDuration { get; set; } : TimeSpan  
+ IsFlashing { get; set; } : bool  
+ Update(screen: GameScreen, gameTime: GameTime) : void  
+ Draw(screen: GameScreen, gameTime: GameTime) : void
```

Paragraph:

Namespace: QuizTime

Inherit: Text

Implement: /

Descrizione:

Classe per la realizzazione di un paragrafo di una determinata larghezza.

Paragraph

```
- delimiterChars: char[]  
- textRows: List<string>  
- capacity: int  
- rowWidth: int  
- verticalSpace: int  
  
+ Paragraph(text: string, font: SpriteFont)  
+ Paragraph(index: int, text: string, font: SpriteFont)  
+ TextContents { get; set; } : string  
+ TextRows { get; } : List<string>  
+ DelimiterChars { get; set; } : char[]  
+ Capacity { get; set; } : int  
+ RowWidth { get; set; } : int  
+ VerticalSpace { get; set; } : int  
+ Update(screen: GameScreen, gameTime: GameTime) : void  
+ Draw(screen: GameScreen, gameTime: GameTime) : void  
+ Height() : int  
+ Width() : int  
- CreateParagraph() : void  
+ InitializeTextRows() : void  
- MisureTextRowLength() : float
```

TextSelectable:

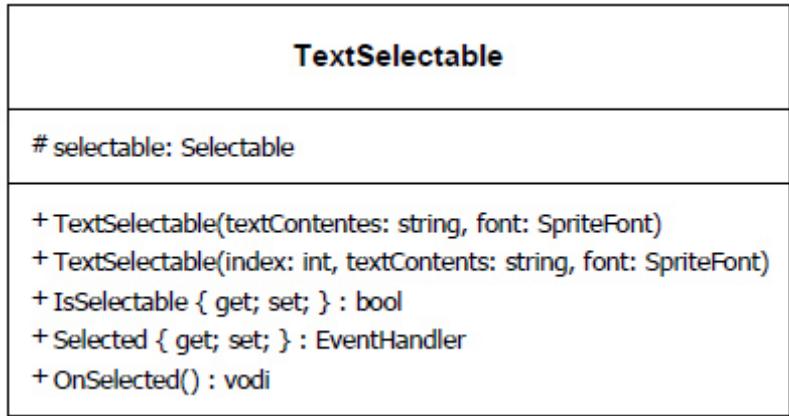
Namespace: QuizTime

Inherit: Text

Implement: /

Descrizione:

Classe per la realizzazione di testo che risponde all'input dell'utente.



Classi ereditarie dirette di TextSelectable

TextButton:

Namespace: QuizTime

Inherit: TextSelectable

Implement: /

Descrizione:

Classe per la realizzazione di un bottone con testo interno, che risponda all'input dell'utente.

TextButton

```
- buttonTexture: Texture2D  
- alphaTexture: float  
- alphaText: float  
  
+ TextButton(textContents: string, font: SpriteFont, buttonTexture: Texture2D)  
+ TextButton(index: int, textContents: string, font: SpriteFont, buttonTexture: Texture2D)  
+ ButtonTexture { get; set; } : Texture2D  
+ AlphaTexture { get; set; } : float  
+ AlphaText { get; set; } : float  
+ AlphaChannel { get; set; } : float  
+ Update(screen: GameScreen, gameTime: GameTime) : void  
+ Draw(screen: GameScreen, gameTime: GameTime) : void  
+ Height(screen: GameScreen) : int  
+ Width(screen: GameScreen) : int  
- getTextPosition(screen: GameScreen) : Vector2
```

AnswerTextButton:

Namespace: QuizTime

Inherit: TextSelectable

Implement: /

Descrizione:

Classe per la realizzazione del bottone delle risposte.

AnswerTextButton

```
- neutralTexture: Texture2D  
- successTexture: Texture2D  
- failureTexture: Texture2D  
- currentTexture: Texture2D  
- horizontalSpace: int  
  
+ AnswerTextButton(textContents: string, font: SpriteFont, failureTexture: Texture2D)  
+ AnswerTextButton(index: int, textContents: string, font: SpriteFont, failureTexture: Texture2D)  
+ Update(screen: GameScreen, gameTime: GameTime) : void  
+ Draw(screen: GameScreen, gameTime: GameTime) : void  
+ Height(screen: GameScreen) : int  
+ Width(screen: GameScreen) : int  
+ SetNeutral() : void  
+ SetSuccess() : void  
+ SetFailure() : void  
- getTexturePosition(screen: GameScreen) : Vector2
```

Classi ereditarie dirette di Image

ImageSelectable:

Namespace: QuizTime

Inherit: Image

Implement: /

Descrizione:

Classe per la realizzazione di immagini che rispondono all'input dell'utente.

ImageSelectable

```
- selectable: Selectable  
  
+ ImageSelectable(imageContents: Texture2D)  
+ ImageSelectable(index: int, imageContents: Texture2D)  
+ IsSelectable { get; set; } : bool  
+ Selected { get; set; } : EventHandler  
+ OnSelected() : void
```

2.3 Prototipi Interfaccia Grafica

Nota:

Le frecce rosse indicano che gli elementi all'avvio della schermata si spostano verso la direzione indicata.

Main Menu screen



All'avvio dell'applicazione viene visualizzata la seguente schermata.

Descrizione:

All'avvio della schermata i pulsanti “Play” e “Highscore” escono dal lato destro dello schermo verso il lato sinistro, l'animazione termina quando raggiungono la loro posizione finale.

Gli elementi con cui l'utente può interagire sono:

- Play → Visualizza la schermata Livelli per la scelta del livello
- Highscore → Visualizza la schermata Lista Punteggi

Highscore screen

Place	Player Name	Score
string	string	string

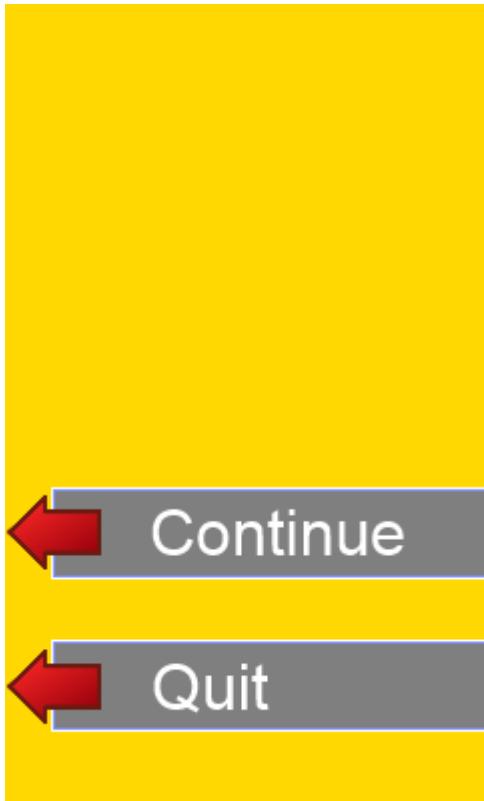
Viene visualizzata quando l'utente seleziona il riquadro “Highscore” dal MainMenu.

Descrizione:

All'avvio della schermata le barre laterali sono posizionate al centro della schermata e si spostano verso i lati corrispondenti della schermata.

Non ci sono elementi selezionabili.

Pause screen



Viene visualizzata quando l'utente sospende una partita.

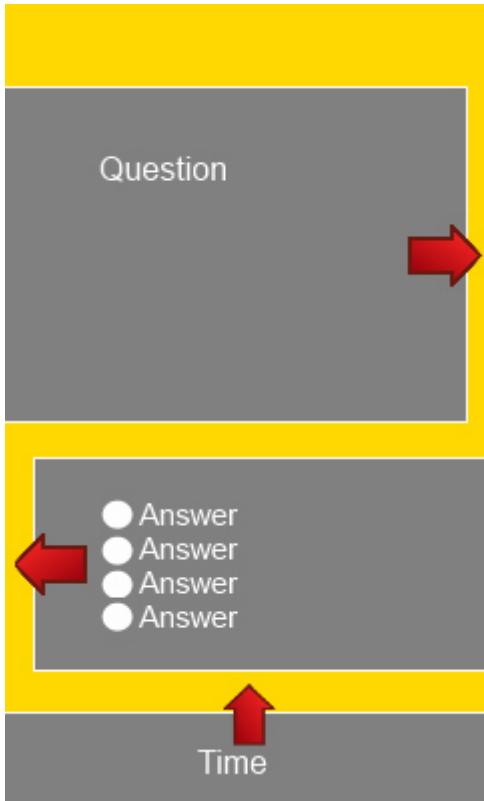
Descrizione:

All'avvio della schermata i pulsanti “Continue” e “Quit” escono dal lato destro dello schermo verso il lato sinistro, l'animazione termina quando raggiungono la loro posizione finale.

Gli elementi selezionabili sono:

- Play → Visualizza la schermata Livelli per la scelta del livello
- Highscore → Visualizza la schermata Lista Punteggi

Gameplay screen



Viene visualizzata all'avvio di una nuova partita.

Descrizione:

All'avvio della schermata:

- la Question-bar esce lateralmente da sinistra verso destra per tutta la sua lunghezza.
- la Answers-bar esce lateralmente da sinistra verso destra per tutta la sua lunghezza.
- la Time-bar esce lateralmente dal basso verso l'alto per tutta la sua lunghezza.

Gli elementi selezionabili sono:

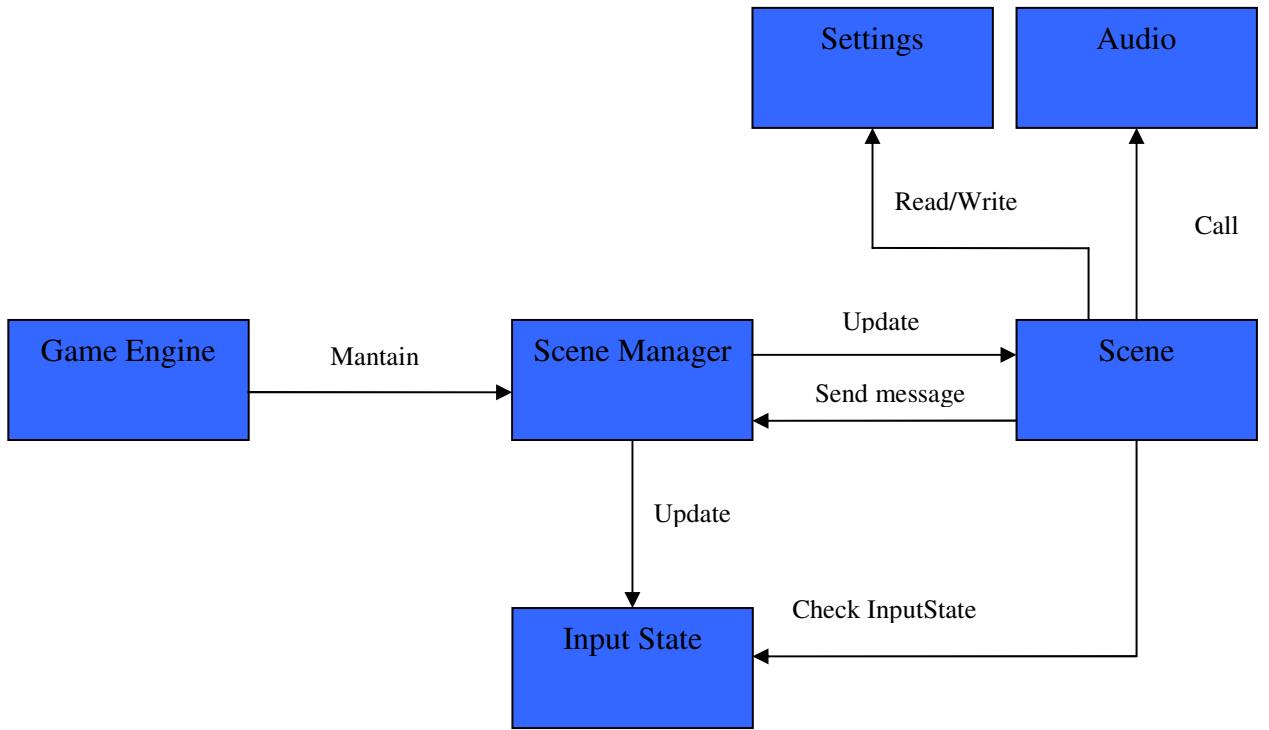
- Answer → Conferma la risposta.

3 Seconda Applicazione Zone

3.1 Progettazione Architetturale

3.1.1 Organizzazione del Sistema

Figura 3.1 Architettura del sistema dell'applicazione Zone



Sottosistemi

Breve descrizione dei sottosistemi di cui è composto il sistema dell'applicazione Zone in Figura 3.1.

GameEngine: Fornisce l'inizializzazione, la logica di gioco e il codice di rendering di base per un dispositivo grafico. Si occupa della creazione e mantenimento di uno Scene Manager.

SceneManager: Gestisce il ciclo di vita delle scene, e dell'avanzamento del loro stato.

Scene: Crea e gestisce gli elementi di una scena su schermo.

InputState: Legge l'input, tenendo traccia dello stato corrente e precedente del dispositivo di input.

Audio: Gestisce tutti i suoni.

Settings: Mantiene i dati delle impostazioni e carica i dati di configurazione.

3.1.2 Scomposizione Modulare

Scomponiamo i sottosistemi in moduli usando una scomposizione orientata agli oggetti.

Con questo approccio i moduli dei sottosistemi verranno rappresentati come oggetti, che richiedono servizi offerti da altri oggetti.

Usiamo i diagrammi dei casi d'uso in UML per mostrare le classi di sistema.

3.1.2.1 Moduli dei Sottosistemi

Insieme dei moduli o delle classi di oggetti individuate nei sottosistemi.

Sottosistema GameEngine

Classi:

- Microsoft.Xna.Framework.Game
- Zone
 - GraphicDeviceManager

Sottosistema SceneManager

Classi:

- ScreenManager

Sottosistema InputState:

Classi:

- InputState

Sottosistema Settings:

Classi:

- SettingsManager

Sottosistema Audio:

Classi:

- AudioManager

Sottosistema Scene:

Classi:

- GameScreen
 - BackgroundScreen
 - MenuScreen
 - BasicMenu
 - OptionsScreen
 - MainMenuScreen
 - PauseScreen
 - HighScoreScreen
 - CreditsScreen
 - GameplayScreen
 - IntroductionScreen
 - LevelScreen
 - LevelTextBox
 - LoadingScreen
 - TextBoxScreen

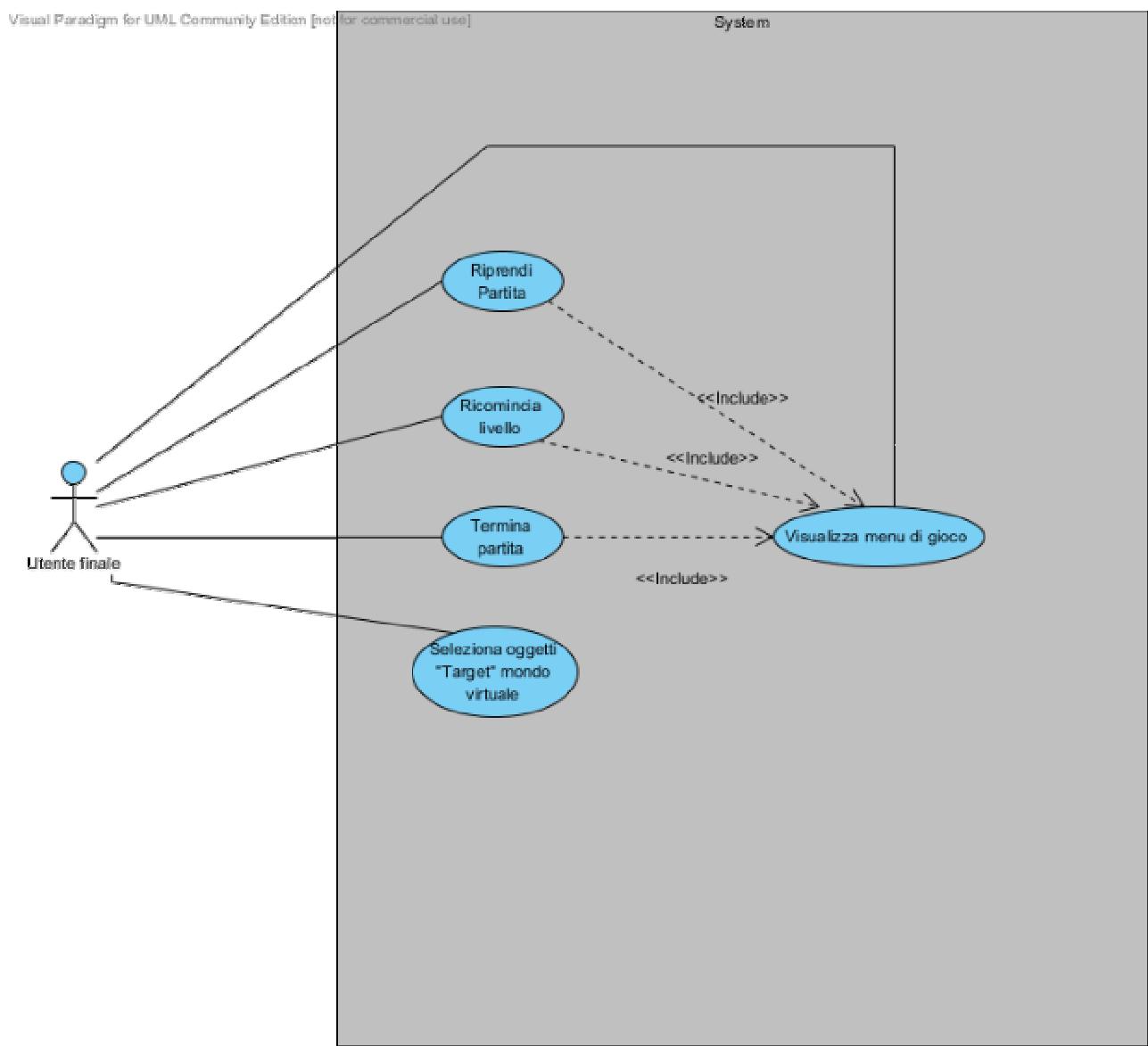
3.2 Progettazione orientata agli oggetti

3.2.1 Diagramma dei casi d'uso

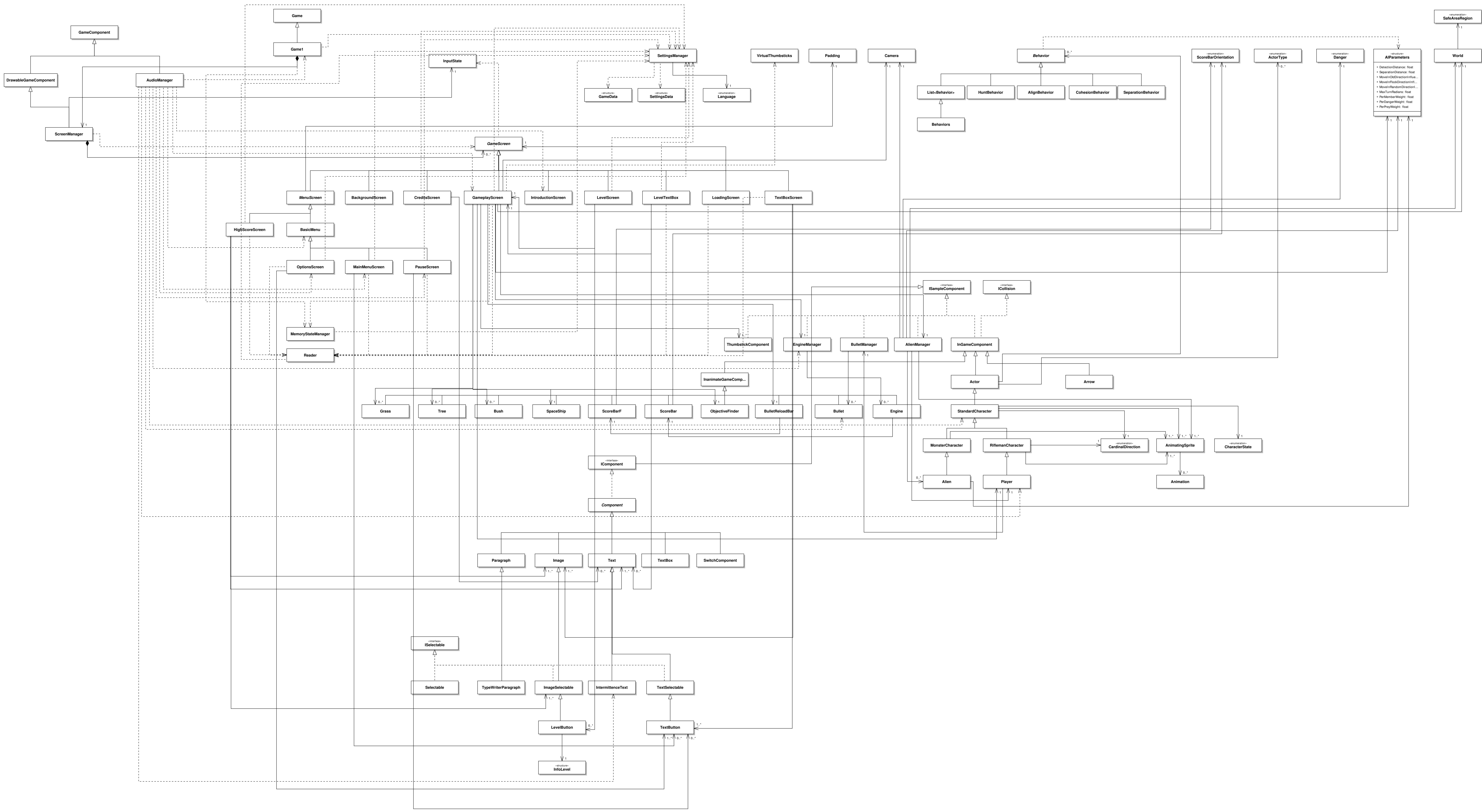
Diagramma Casi d'uso Menu Principale



Diagramma casi d'uso Livello di gioco



3.2.2 Diagramma delle classi



3.2.2.1 Descrizione delle interfacce

ISampleComponent:

Namespace: ZoneGame

Inherit: /

Descrizione:

Interfaccia dalle classi che devono aggiornare il proprio stato ed essere disegnati su schermo.



l’interfaccia ISampleComponent è estesa dalle interfacce:

- IComponent

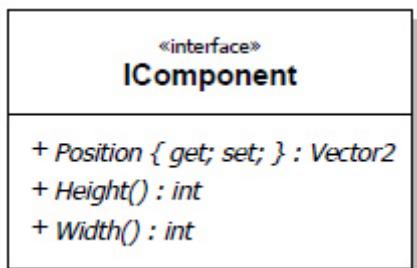
IComponent:

Namespace: ZoneGame

Inherit: ISampleComponent

Descrizione:

Interfaccia implementata dalle classi che oltre a dover implementare un metodo per disegnare e aggiornare lo stato, è richiesta la posizione e le dimensioni(altezza, larghezza).



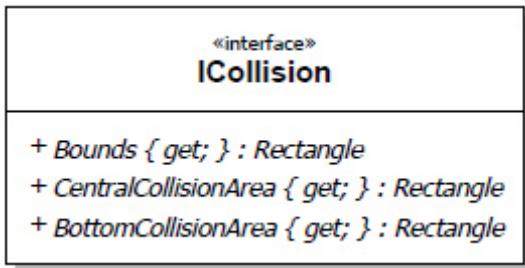
ICollision:

Namespace: ZoneGame

Inherit: /

Descrizione:

Interfaccia da implementare per la gestione delle collisioni.



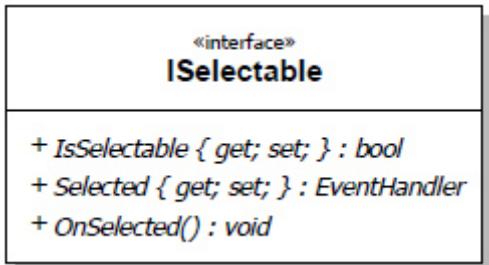
ISelectable:

Namespace: ZoneGame

Inherit: /

Descrizione:

Interfaccia implementata dalle classi che devono gestire gli input dati dall'utente.



3.2.2.2 Descrizione delle strutture dati

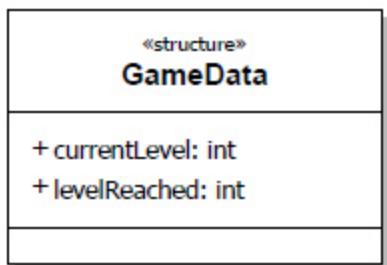
GameData:

Namespace: ZoneGame

Descrizione:

Struttura adibita a contenere le variabili per inizializzare i campi della classe GameplayScreen.

Utilizzata per salvare e caricare il valore dei campi della classe GameplayScreen.



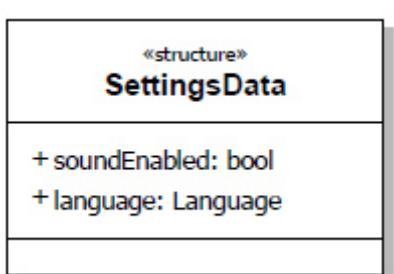
SettingsData:

Namespace: ZoneGame

Descrizione:

Struttura adibita a contenere le variabili per inizializzare i campi della classe SettingsManager.

Utilizzata per salvare e caricare il valore dei campi della classe SettingsManager.

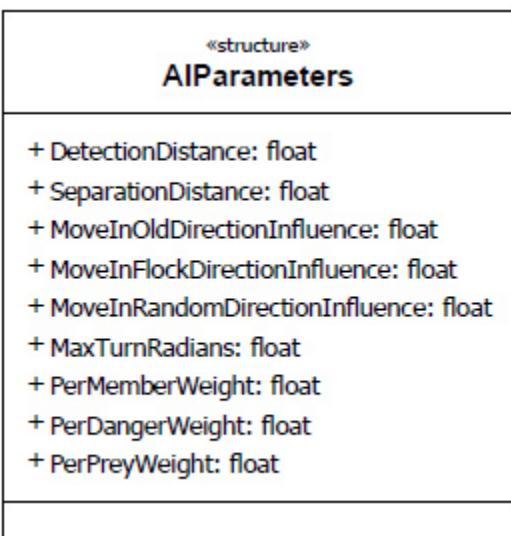


AIParameters:

Namespace: ZoneGame

Descrizione:

Struttura che contiene i campi usati per realizzare il comportamento di Flocking.

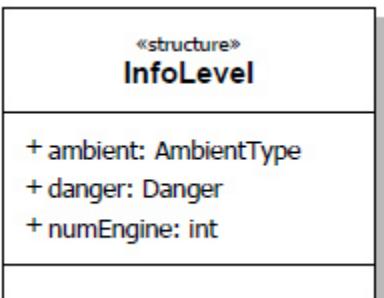


InfoLevel:

Namespace: ZoneGame

Descrizione:

Struttura che contiene le informazioni generali di un livello.



3.2.2.3 Descrizione degli enumeratori

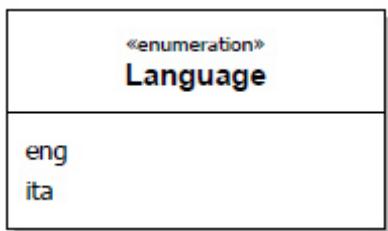
Enumeratori

Language:

Namespace: ZoneGame

Descrizione:

Identifica il set di nomi usati per distinguere le diverse lingue.

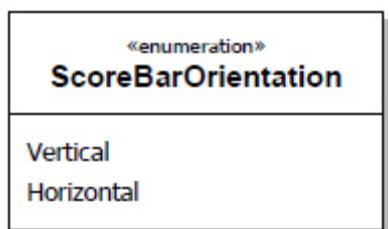


ScoreBarOrientation:

Namespace: ZoneGame

Descrizione:

Identifica il set di nomi usati per definire l'orientamento di un score bar.

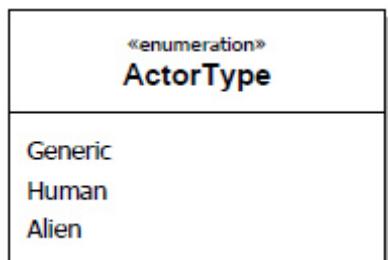


ActorType:

Namespace: ZoneGame

Descrizione:

Identifica il set di nomi usati per distinguere i diversi attori che operano nella mappa.



CardinalDirection

Namespace: ZoneGame

Descrizione:

Identifica il set di nomi usati per indicare la direzione cardinale dei personaggi nella mappa.

«enumeration»	CardinalDirection
North	
NorthNortheast	
NorthEast	
EastNortheast	
East	
EastSoutheast	
SouthEast	
SouthSoutheast	
South	
SouthSouthwest	
SouthWest	
WestSouthwest	
West	
WestNorthwest	
NorthWest	
NorthNorthwest	

Danger:

Namespace: ZoneGame

Descrizione:

Identifica il set di nomi usati per distinguere i diversi gradi di difficoltà.

«enumeration»	Danger
Low	
Medium	
High	

SafeAreaRegion:

Namespace: ZoneGame

Descrizione:

Identifica il set di nomi usati per definire la posizione iniziale sulla mappa della safeArea.



3.2.2.4 Descrizione delle classi

Game

Namespace: Microsoft.Xna.Framework

Inherit: /

Implement: IDisposable

Descrizione:

Classe contenuta nella libreria XNA Framework, fornisce l'inizializzazione, la logica di gioco e il codice di rendering di base per un dispositivo grafico.

GameComponent

Namespace: Microsoft.Xna.Framework

Inherit: IGameComponent

Implement: IUpdateable, IDisposable

Descrizione:

Classe base per tutti i componenti di gioco di XNA Framework. Un componente di gioco che estende questa classe dispone di metodi di aggiornamento e inizializzazione chiamati dai metodi Game.Initialize, Game.Update qualora sia stato registrato con in metodo Game.Component.Add.

DrawableGameComponent

Namespace: Microsoft.Xna.Framework

Inherit: GameComponent

Implement: IDrawable

Descrizione:

Componente del gioco a cui viene inviata una notifica quando deve disegnarsi automaticamente.

Game1

Namespace: ZoneGame

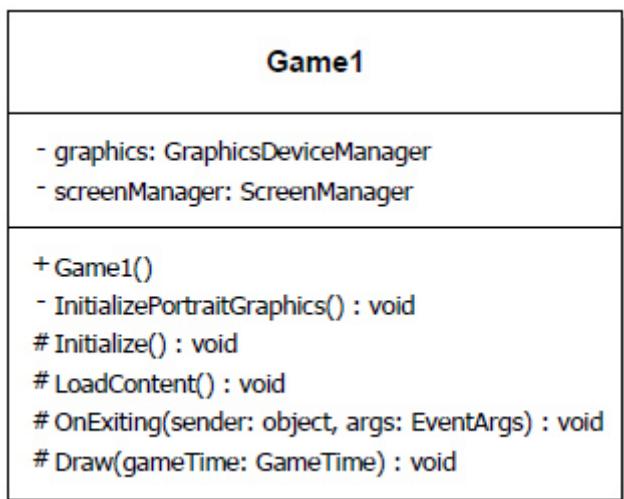
Inherit: Game

Implement: /

Descrizione:

Eredita la classe Game.

Contiene un'istanza dell'oggetto GraphicDeviceManager che consente di configurare e gestire il dispositivo grafico, e un'istanza dell'oggetto ScreenManager usato per gestire le scene.



ScreenManager

Namespace: ZoneGame

Inherit: DrawableGameComponent

Implement: /

Descrizione:

Contiene un elenco di oggetti di tipo GameScreen, di ogni GameScreen nella lista viene chiamato il metodo di Update a cui viene passato la variabile booleana coveredByOtherScreen che serve a stabilire se lo screen debba essere nascosto e lasciare posto ad un altro screen. Inoltre richiama il metodo HandleInput dello screen attivo per la gestione dell'input, se non ci sono altri screen che hanno il focus.

ScreenManager	
- screens: List<GameScreen>	
- screensToUpdate: List<GameScreen>	
- input: InputState	
- spriteBatch: SpriteBatch	
- font: SpriteFont	
- blankTexture: Texture2D	
- siInitialized: bool	
+ SpriteBatch { get; } : SpriteBatch	
+ Font { get; } : SpriteFont	
+ BlankTexture { get; } : Texture2D	
+ ScreenManager(game: Game)	
+ Initialize() : void	
# LoadContent() : void	
# UnloadContent() : void	
+ Update(gameTime: GameTime) : void	
+ Draw(gaemTime: GameTime) : void	
+ AddScreen(screen: GameScreen) : void	
+ RemoveScreen(screen: GameScreen) : void	
+ GetScreens() : GameScreen[]	
+ FadeBackBufferToBlack(alpha: float) : void	
+ SerializeState() : void	
+ DeserializeState() : bool	
- DeleteState(storage: IsolatedStorageFile) : void	

InputState

Namespace: ZoneGame

Inherit: /

Implement: /

Descrizione:

Questa classe legge gli input tenendo traccia del corrente e precedente stato dell'input, è utile per rilevare gli input dello smartphone, ma può essere esteso anche a dispositivi come tastiera, mouse e controller.

InputState

```
+ MaxInputs: int = 4
+ CurrentGamePadStates: GamePadState[]
+ LastGamePadStates: KeyboardState[]
+ GamePadWasConnected: bool[]
+ TouchState: TouchCollection
+ Gestures: List<GestureSample> = new List<GestureSample>()

+ InputState()
+ Update() : void
# CheckButton(button: Button, controllingPlayer: PlayerIndex?) : bool
+ isNewButtonPress(button: Buttons, controllingPlayer: PlayerIndex?, out playerIndex: PlayerIndex) : bool
+ IsMenuSelect() : bool
+ IsMenuCancel() : bool
+ IsPauseGame() : bool
```

SettingsManager

Namespace: ZoneGame

Inherit: /

Implement: /

Descrizione:

Mantiene l'insieme delle variabili di configurazione.

SettingsManager

```
- maxLevel: int
- levelReached: int
- currentLevel: int
- isResuming: bool
- isSave: bool
- language: Language

+ MaxLevel { get; } : int
+ LevelReached { get; set; } : int
+ CurrentLevel { get; set; } : int
+ IsResuming { get; set; } : bool
+ IsSave { get; set; } : bool
+ Language { get; set; } : Language
+ LoadMaxLevel() : int
```

AudioManager

Namespace: ZoneGame

Inherit: GameComponent

Implement: /

Descrizione:

Carica tutti i suoni e le musiche, e implementa i metodi per la loro gestione.
Per questa classe è stato usato il pattern Singleton.

AudioManager	
- <code>isActive: bool</code>	- <code>isInitialized: bool</code>
- <code>audioManager: AudioManager = null</code>	- <code>soundAssetLocation: string = "Sounds/"</code>
- <code>soundBank: Dictionary<string, SoundEffectInstance></code>	- <code>musicBank: Dictionary<string, Song></code>
- <code>AudioManager(game: Game)</code>	+ <code>Instance { get; } : AudioManager</code>
+ <code>IsInitialized { get; } : bool</code>	+ <code>IsActive { get; set; } : bool</code>
+ <code>Initialize(game: Game) : void</code>	+ <code>LoadSound(contentName: string, alias: string) : void</code>
+ <code>LoadSong(contentName: string, alias: string) : void</code>	+ <code>LoadSongs() : void</code>
+ <code>LoadMusic() : void</code>	+ <code>PlaySound(soundName: string) : void</code>
+ <code>this[string soundName] { get; } : SoundEffectInstance</code>	+ <code>PlaySound(soundName: string, isLooped: bool) : void</code>
+ <code>PlaySound(soundName: string, isLooped: bool, volume: float) : void</code>	+ <code>StopSound(soundName: string) : void</code>
+ <code>StopSounds() : void</code>	+ <code>PauseResumeSounds(resumeSounds: bool) : void</code>
+ <code>PlayMusic(musicSoundName: string) : void</code>	+ <code>PlayMusic() : void</code>
+ <code>StopMusic() : void</code>	+ <code>Enable(state: bool) : void</code>
# <code>Dispose(disposing: bool) : void</code>	

GameScreen

Namespace: ZoneGame

Inherit: /

Implement: /

Descrizione:

E' la classe astratta che rappresenta una scena su schermo.

GameScreen

```
# contentDynamic: ContentManager
# LanguageDefinitions: Dictionary<string,string>
- isPopup: bool = false
- screenState: ScreenState = ScreenState.TransitionOn
- isExiting: bool = false
- otherScreenHasFocus: bool
- screenManager: ScreenManager
- enabledGesture: GestureType = GestureType.None
- isSerializable: bool = true

+ isPopup { get; set; } : bool
+ ScreenState { get; set; } : ScreenState
+ isExiting { get; set; } : bool
+ isActive { get; } : bool
+ ScreenManager { get; set; } : ScreenManager
+ EnabledGestures { get; set; } : GestureType
+ isSerializable { get; set; } : bool
+ Initialize() : void
+ LoadContent() : void
+ UnloadContent() : void
+ Update(gameTime: GameTime, otherScreenHasFocus: bool, coveredByOtherScreen: bool) : void
+ Draw(gameTime: GameTime) : void
+ HandleInput(input: InputState) : void
+ Serialize(stream: Stream) : void
+ Deserialize(stream: Stream) : void
+ ExitScreen() : void
+ LoadAssets() : void
+ Load<T>(assetName: string) : T
# ReplaceForwardScreens(screens: List<GameScreen>) : void
# ReplaceAllScreens(screens: List<GameScreen>) : void
```

Classi Ereditarie di GameScene

Classi Ereditarie dirette di GameScreen

La classe GameScreen è ereditata dalle classi:

- MenuScreen
- BackgroundScreen
- CreditsScreen

- GameplayScreen
- IntroductionScreen
- LevelScreen
- LevelTextBox
- LoadingScreen
- TextBoxScreen

MenuScreen:

Namespace: ZoneGame

Inherit: GameScreen

Implement: /

Descrizione:

Classe astratta che estende GameScreen. Possiede un elenco di oggetti di tipo Entry, di cui gestisce gli eventi.

MenuScreen	
- components: List<IComponent>	# padding: Padding
+ MenuScreen()	
# Components { get; } : IList<IComponent>	
+ LoadContent() : void	
+ HandleInput(input: InputState) : void	
+ Update(gameTime: GameTime, otherScreenHasFocus: bool, coveredByOtherScreen: bool) : void	
+ Draw(gameTime: GameTime) : void	
# UpdateComponentsLocations() : void	
# GetEntryHitBounds(component: IComponent) : Rectangle	
# OnSelectEntry(entryIndex: int) : void	
# OnCancel() : void	

La classe MenuScreen è estesa dalle seguenti classi:

- HighScoreScreen
- BasicMenu

BackgroundScreen:

Namespace: ZoneGame

Inherit: GameScreen

Implement: /

Descrizione:

Rappresenta il background sullo schermo, non può gestire gli input e consente agli altri screen di sovrapporsi e gestire gli input

BackgroundScreen

```
- backgroundTexture: Texture2D  
- backgroundName: string  
  
+ BackgroundScreen(backgroundName: string)  
+ LoadContent() : void  
+ Update(gameTime: GameTime, otherScreenHasFocus: bool, coveredByOtherScreen: bool) : void  
+ Draw(gameTime: GameTime) : void
```

CreditsScreen:

Namespace: ZoneGame

Inherit: GameScreen

Implement: /

Descrizione:

Rappresenta la schermata dei crediti.

CreditsScreen

```
- CreditsDocPath: string = "Content/Documents/CreditsContent.xml"  
- font14px: SpriteFont  
- position: Vector2  
- speed: Vector2  
- actorSpace: int  
- actornamespace: int  
- nameSpace: int  
- creditsHeight: int  
- isPlay: bool  
- creditsStartPosition: Vector2  
  
+ CreditsScreen()  
+ LoadContent() : void  
+ HandleInput(input: InputState) : void  
+ Update(gameTime: GameTime, otherScreenHasFocus: bool, coveredByOtherScreen: bool) : void  
- InitializeCreditsPosition(position: Vector2, font: SpriteFont) : void  
- GetHeight(font: SpriteFont) : int  
# OnCancel() : void
```

GameplayScreen:

Namespace: ZoneGame

Inherit: GameScreen

Implement: /

Descrizione:

E' la classe che gestisce la schermata di gioco.

GameplayScreen	
- IsActive: bool	
- random: Random	
- queryList: List<List<Question>>	
- outerListIndex: int	
- innerListIndex: int	
- currentRightAnswer: string	
- maxQuestionPerList: int	
- isShowComponents: bool	
- userTapToStar: bool	
- isUserWon: bool	
- gameEnded: bool	
- userTapToExit: bool	
- hasAnswered: bool	
- insertInHighScore: bool	
- nextQuery: bool	
- isLifeLost: bool	
- isPlaySound: bool	
- isAlarmClock: bool	
- moveToHighScore: bool	
- rightAnswers: int	
- currentQuestionNumber: int	
- maxQuestionNumber: int	
- questionBar: QuestionBar	
- answerBar: AnswerBar	
- lifeBar: LifeBar	
- clockBar: ClockBar	
- flashingText: FlashingText	
- touchNotice: Text	
- questionParagraph: Paragraph	
- answerButtons: List<AnswerTextButton>	
- timeToWait: TimeSpan	
- timeElapsed: TimeSpan	
- flashingDuration: TimeSpan	
- maxClockTime: TimeSpan	
- isResuming: bool	
+ GameplayScreen()	
+ IsActive { get; set; } : bool	
+ RightAnswers { get; } : int	
- IsOpenComponentsBar { get; } : bool	
- IsCloseComponentsBar { get; } : bool	
- ComponentsBarIsOpened { get; } : bool	
- ComponentsBarIsClosed { get; } : bool	
- IsOpenedComponentsGrill { get; } : bool	
- ComponentsGrillIsOpened { get; } : bool	
- ComponentsGrillIsClosed { get; } : bool	
+ LoadContent() : void	
+ LoadAssets() : void	
+ LoadAssets() : void	
- LoadQuestionsFromXML() : void	
- LoadTextures() : void	
- CreateGameComponents() : void	
+ InitializeQueryComponents() : void	
+ InitializeAnimationFields() : void	
+ HandleInput(input: InputState) : void	
+ Update(gameTime: GameTime, otherScreenHasFocus: bool, coveredByOtherScreen: bool) : void	
+ Draw(gameTime: GameTime) : void	
# GetEntryHitBounds(component: IComponent) : Rectangle	
- CheckIfCurrentGameFinished() : bool	
- CheckIsInHighScore() : bool	
- DrawGameEndIfNecessary() : void	
- PauseCurrentGame() : void	

IntroductionScreen:

Namespace: ZoneGame

Inherit: GameScreen

Implement: /

Descrizione:

Classe per la realizzazione della schermata introduttiva.

IntroductionScreen

- blankTexture: Texture2D
- badmoodTexture: Texture2D
- blankDimension: Rectangle
- badmoodPosition: Vector2
- fon14pxt: SpriteFont
- alphaChannel: float
- transAlpha: float
- badmoodScale: float
- isLogoVisible: bool
- isScreenCreated: bool
- badmoodColor: Color
- timeToWait: TimeSpan
- timeElapsed: TimeSpan

- + LoadContent() : void
- + Update(gameTime: GameTime, otherScreenHasFocus: bool, coveredByOtherScreen: bool) : void
- + Draw(gameTime: GameTime) : void

LevelScreen:

Namespace: ZoneGame

Inherit: GameScreen

Implement: /

Descrizione:

Classe che realizza la schermata di selezione del livello di gioco.

LevelScreen	
- backgroundText: Texture2D	
- levelButtonText: Texture2D	
- textFont: SpriteFont	
- quartzFont: SpriteFont	
- spriteBatch: SpriteBatch	
- screen: GameplayScreen	
- maxLevel: int	
- currentLevelSelected: int	
- levelButtons: List<LevelButton>	
+ LevelScreen(screen: GameplayScreen)	
+ LoadContent() : void	
+ HandleInput(input: InputState) : void	
+ Draw(gameTime: GameTime) : void	
- CreateInfoLevel(elemLevel: XElement) : InfoLevel	
- CompareLevels(b1: LevelButton, b2: LevelButton) : int	
- UpdateLevelButtonsPosition() : void	
- OnLevelButtonSelected(sender: object, e: EventArgs) : void	
# onCancel() : void	

LevelTextBox:

Namespace: ZoneGame

Inherit: GameScreen

Implement: /

Descrizione:

Classe per la realizzazione di una finestra di pop-up usata nella schermata della scelta del livello per la visualizzazione delle informazioni del livello selezionato.

LevelTextBox

```
- info: InfoLevel  
- levelNum: int  
- isUnlocked: bool  
- screen: GameplayScreen  
- texts: List<Text> = new List<Text>()  
  
+ LevelTextBox(isPopup: bool, info: InfoLevel, levelNum: int, isUnlocked: bool, screen: Gameplay)  
+ LoadContent() : void  
+ Draw(gameTime: GameTime) : void  
# OnBackButtonSelected(sender: object, e: EventArgs) : void  
# OnConfirmButtonSelected(sender: object, e: EventArgs) : void  
# CreateTextContent(strArray: String []) : void  
# UpdateTextPosition(startPos: Vector2) : void  
- GetAmbientTypeString(aType: AmbientType) : string  
- GetDangerTypeString(dType: Danget) : string
```

LoadingScreen:

Namespace: ZoneGame

Inherit: GameScreen

Implement: /

Descrizione:

Indica la schermata di caricamento, viene lanciata prima del GameplayScreen per consentire il caricando dei contenuti di gioco.

LoadingScreen

- LoadingText: Text
- DotText: Text
- font: SpriteFont
- isLoading: bool
- isReady: bool
- screen: GameScreen
- loadingThread: Thread

+ LoadingScreen(screen: GameScreen)
+ LoadContent() : void
+ UnloadContent() : void
+ HandleInput(input: InputState) : void
+ Update(gameTime: GameTime, otherScreenHasFocus: bool, coveredByOtherScreen: bool) : void
+ Draw(gameTime: GameTime) : void
- LoadResources() : void
PlayScreen() : void
OnCancel() : void

TextBoxScreen:

Namespace: ZoneGame

Inherit: GameScreen

Implement: /

Descrizione:

Classe per la realizzazione di un contenitore di bottoni.

TextBoxScreen

```
# topTexture: Texture2D
# bottomTexture: Texture2D
# buttonTexture: Texture2D
# blankTexture: Texture2D
# font: SpriteFont
# textFont: SpriteFont
# spriteBatch: SpriteBatch
# topBar: Image
# bottomBar: Image
# backButton: TextButton
# confirmButton: TextButton
# buttons: List<TextButton>
# dimension: Rectangle

+ TextBoxScreen(isPopup: bool)
+ LoadContent() : void
+ HandleInput(input: InputState) : void
+ Draw(gameTime: GameTime) : void
# UpdateComponentsPosition() : void
# UpdateComponentsPosition() : void
# OnBlackButtonSelected(sender: object, e: EventArgs) : void
# OnConfirmButtonSelected(sender: object, e: EventArgs) : void
# onCancel() : void
```

Classi Ereditarie dirette di MenuScreen

HighScoreScreen:

Namespace: ZoneGame

Inherit: MenuScreen

Implement: /

Descrizione:

classe per la gestione della schermata dei punteggi.

HighScoreScreen

- HighScoreDataDestination: string = "HighscoreData.sav"
- currentLevel: int
- totLevels: int
- highscoorePlaces: int = 5
- + highScore: List<KeyValuePair<string,int>>[]
- numberPlaceMapping: Dictionary<int, string>
- title: Text
- level: Text
- page: Text
- topBar: Image
- bottomBar: Image
- levelBar: Image
- prevButton: ImageSelectable
- nextButton: ImageSelectable
- glassScreenDimension: Rectangle

- + HighScoreScreen()
- + LoadContent() : void
- + HandleInput(input: InputState) : void
- + Update(gameTime: GameTime, otherScreenHasFocus: bool, coveredByOtherScreen: bool) : void
- + Draw(gameTime: GameTime) : void
- AnimateScoreBar() : void
- AnimateLevelBar() : void
- AnimateGlassScreen() : void
- UpdateTextContent() : void
- UpdateTextPosition() : void
- UpdateButtonPosition() : void
- DrawHighScore(spriteBatch: SpriteBatch) : void
- Exit() : void
- + IsInHighscores(score: int, level: int) : bool
- + PutHighScore(playerName: string, score: int, level: int) : void
- OrderGameScore(level: int) : void
- + CompareScores(score1: KeyValuePair<string, int>, score2: KeyValuePair<string, int>) : int
- + SaveHighscore() : void
- + LoadHighscores() : void
- GetPlaceString(number: int) : string
- initializeMapping() : void
- OnPreviousButtonSelected(sender: object, e: EventArgs) : void
- OnNextButtonSelected(sender: object, e: EventArgs) : void
- InitializeScore() : List<KeyValuePair<string, int>> []

BasicMenu:

Namespace: ZoneGame

Inherit: MenuScreen

Implement: /

Descrizione:

Classe che costruisce le linee guida per la realizzazione delle schermate del menu principale. Ereditata dalle classi MainMenuScreen, PauseScreen, OptionScreen per la realizzazione della schermata del main menu del menu di pausa.

BasicMenu

```
- verticalComponents: List<IComponent>
- freeComponents: List<IComponent>
# textBlockDistance: int
# sideBarAnimationStep: int
# glassScreenAnimationStep: int
# blankTexture: Texture2D
# buttonMenuTexture: Texture2D
# topSideBarTexture: Texture2D
# bottomSideBarTexture: Texture2D
# menuFont: SpriteFont
# textColor: Color
# alphaChannel: float
# title: Text
# topSideBar: Image
# bottomSideBar: Image
- animateSideBar: bool
- sideBarInTransition: bool
- sideBarHitFinalPosition: bool
- glassScreenInTransition: bool
- glassScreenHitFinalPosition: bool
- topSideBarClosedPosition: Vector2
- topSideBarOpenedPosition: Vector2
- bottomSideBarClosedPosition: Vector2
- bottomSideBarOpenedPosition: Vector2
- glassScreenDimension: Rectangle

+ BasicMenu()
+ IsPlayMusic { get; set; } : bool
# Title { get; } : IComponent
# MenuFont { get; set; } : SpriteFont
+ LoadContent() : void
+ HandleInput(input: InputState) : void
+ Update(gameTime: GameTime, otherScreenHasFocus: bool, coveredByOtherScreen: bool) : void
+ Draw(gameTime: GameTime) : void
- AnimateSideBar() : void
- AnimateGlassScreen() : void
# UpdateComponentsLocation() : void
# GetEntryHitBounds(component: IComponent) : Rectangle
# AddVerticalTextButton(textButton: TextButton) : void
- AddVerticalComponent(component: IComponent) : void
# PlayMusic(play: bool, strSong: string) : void
```

Classi Ereditarie dirette di BasicMenu

OptionsScreen:

Namespace: ZoneGame

Inherit: BasicMenu

Implement: /

Descrizione:

Classe per la realizzazione della schermata delle impostazioni di gioco.

OptionsScreen

- languages: Dictionary<Language, String>
- SoundEntry: TextButton
- LanguageEntry: TextButton

- + LoadContent() : void
- SoundEntrySelected(sender: object, e: EventArgs) : void
- LanguageEntrySelected(sender: object, e: EventArgs) : void
- InitializeButtons() : void
- SetEntryText() : void
- # OnCancel() : void

MainMenuItem:

Namespace: ZoneGame

Inherit: BasicMenu

Implement: /

Descrizione:

Classe per la realizzazione della schermata del menu principale.

MainMenuItem

- textButtons: List<TextButton>

- + LoadContent() : void
- InitializeButtons() : void
- StartEntrySelected(sender: object, e: EventArgs) : void
- ContinueEntrySelected(sender: object, e: EventArgs) : void
- OptionsEntrySelected(sender: object, e: EventArgs) : void
- HelpEntrySelected(sender: object, e: EventArgs) : void
- HighscoreEntrySelected(sender: object, e: EventArgs) : void
- CreditsEntrySelected(sender: object, e: EventArgs) : void
- TestButtons(text: string) : void
- # OnCancel() : void

PauseScreen:

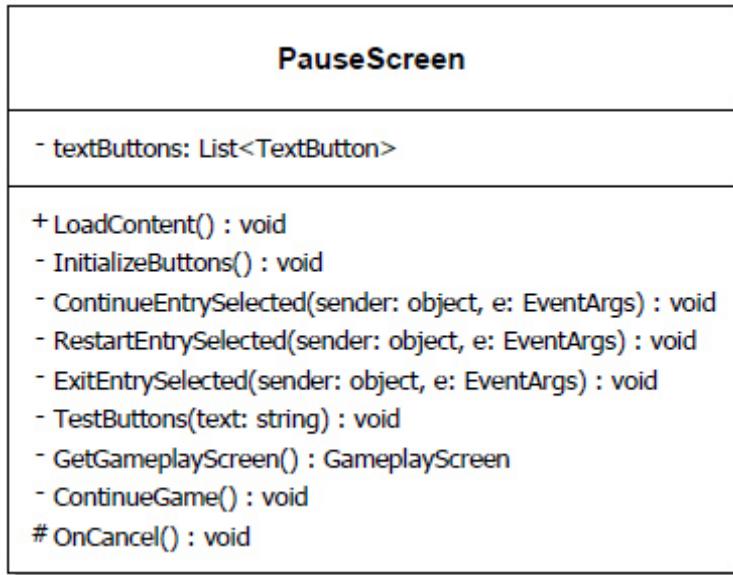
Namespace: ZoneGame

Inherit: BasicMenu

Implement: /

Descrizione:

Classe per la realizzazione del menu di pausa.



Altre Classi

Padding:

Namespace: ZoneGame

Inherit: /

Implement: /

Descrizione: /

Padding

```
+ Padding(top_right_bottom_left: uint)
+ Padding(top_bottom: uint, right_left: uint)
+ Padding(top: uint, right_left: uint, bottom: uint)
+ Padding(top: uint, right: uint, left: uint, bottom: uint)
+ Top { get; set; } : uint
+ Right { get; set; } : uint
+ Bottom { get; set; } : uint
+ Left { get; set; } : uint
+ Zero { get; } : Padding
+ AssignValues(top_right_bottom_left: uint) : void
+ AssignValues(top_bottom: uint, right_left: uint) : void
+ AssignValues(top: uint, right_left: uint, bottom: uint) : void
+ AssignValues(top: uint, right: uint, left: uint, bottom: uint) : void
```

MemoryStateManager:

Namespace: ZoneGame

Inherit: /

Implement: /

Descrizione:

Classe che contiene i metodi per salvare e caricare i dati per inizializzare la classe SettingsManager e i dati delle partite giocate.

MemoryStateManager

```
- GameDataDestination: string = "GameData.sav"
- SettingsDataDestination: string = "SettingsData.sav"

+ GameDataFileExists() : bool
+ SettingsDataFileExists() : bool
+ LoadGameData() : GameData
+ SaveGameData(gamedata: GameData) : void
+ LoadSettingsData() : SettingsData
+ SaveSettingsData(settingsdata: SettingsData) : void
+ SaveCurrentGameState(currentLevel: int) : void
```

Reader:

Namespace: ZoneGame

Inherit: /

Implement: /

Descrizione:

Classe utilizzata per la lettura del documento contenente le definizioni del linguaggio attualmente attivo.

Reader

- LanguageDocPath: string = "Content/Documents/Language.xml"
 - LevelDocPath: string = "Content/Documents/Level.xml"
 - HelpReferencesDocPath: string = "Content/Documents/HelpReferences.xml"
- + [LoadLanguage\(sceneTag: string\) : Dictionary<string,string>](#)

Camera:

Namespace: ZoneGame

Inherit: /

Implement: /

Descrizione:

Classe che ricrea le caratteristiche di una telecamera per navigare all'interno del mondo virtuale.

Camera

- graphicsDevice: GraphicsDevice
 - m_CameraPosition: Vector2
 - visibleArea: Rectangle
 - m_Zoom: float
 - m_Rotation: float
 - m_Transform: Matrix
- + Camera(graphics: GraphicsDevice)
- + Zoom { get; set; } : float
- + Rotation { get; set; } : float
- + Position { get; set; } : Vector2
- + ScreenPosition { get; } : Vector2
- + VisibleArea { get; } : Rectangle
- + ViewingPosition { get; set; } : Vector2
- + ViewingPositionX { get; set; } : float
- + ViewingPositionY { get; set; } : float
- + ViewingWidth { get; set; } : float
- + ViewingHeight { get; set; } : float
- + Move(amount: Vector2) : void
- + Transform() : Matrix

VirtualThumbsticks:

Namespace: ZoneGame

Inherit: /

Implement: /

Descrizione:

Classe per la gestione dei controller.

VirtualThumbsticks	
-	<code>maxThumbstickDistance: float = 60f</code>
-	<code><u>leftPosition: Vector2</u></code>
-	<code><u>rightPosition: Vector2</u></code>
-	<code><u>leftId: int</u></code>
-	<code><u>rightId: int</u></code>
+	<code><u>LeftThumbstickCenter { get; set; } : Vector2?</u></code>
+	<code><u>RightThumbstickCenter { get; set; } : Vector2?</u></code>
+	<code><u>LeftThumbstick { get; } : Vector2</u></code>
+	<code><u>RightThumbstick { get; } : Vector2</u></code>
+	<code><u>Update(input: InputState) : void</u></code>

Behavior:

Namespace: ZoneGame

Inherit: /

Implement: /

Descrizione:

Classe di base per lo sviluppo di una classe atteggiamento.

Behavior
<pre>- actor: Actor # reaction: Vector2 # reacted: bool # Behavior(actor: Actor) + Actor { get; set; } : Actor + Reaction { get; } : Vector2 + Reacted { get; } : bool + Update(otherActor: Actor, aiParams: AIParameters) : void # ResetReaction() : void</pre>

Animation:

Namespace: ZoneGame

Inherit: /

Implement: /

Descrizione:

Classe che rappresenta un'animazione.

Animation
<pre>- name: string - startingFrame: int - endingFrame: int - interval: int - isLoop: bool + Name { get; set; } : string + StartingFrame { get; set; } : int + EndingFrame { get; set; } : int + Interval { get; set; } : int + NewProperty { get; set; } : int</pre>

AnimatingSprite:

Namespace: ZoneGame

Inherit: /

Implement: /

Descrizione:

Classe che gestisce un insieme di animazioni di uno sprite.

AnimatingSprite

- textureName: string
- texture: Texture2D
- textureData: Color[]
- frameDimension: Point
- frameOrigin: Point
- framesPerRow: int
- sourceOffset: Vector2
- animations: List<Animation>
- currentAnimation: Animation
- currentFrame: int
- elapsedTime: float
- sourceRectangle: Rectangle

+ TextureName { get; set; } : string
+ Texture { get; set; } : Texture2D
+ TextureData { get; set; } : Color[]
+ currentSpriteTextureData { get; } : Color[]
+ FrameDimensions { get; set; } : Point
+ FramesPerRow { get; set; } : int
+ SourceOffset { get; set; } : Vector2
+ Animations { get; set; } : List<Animation>
+ this[string animationName] { get; } : Animation
+ AddAnimation(animation: Animation) : bool
+ CurrentFrame { get; } : int
+ StartingFrame { get; } : int
+ EndingFrame { get; } : int
+ CurrentRowFrame { get; } : int
+ SourceRectangle { get; } : Rectangle
+ PlayAnimation(animation: Animation, reset: bool) : void
+ PlayAnimation(index: int, reset: bool) : void
+ PlayAnimation(name: string, reset: bool) : void
+ PlayAnimation(name: string, direction: CardinalDirection, reset: bool) : void
+ ResetAnimation() : void
+ AdvanceToEnd() : void
+ StopAnimation() : void
+ IsPlaybackComplete { get; } : bool
+ UpdateAnimation(elapsedSeconds: float) : void
+ Draw(spriteBatch: SpriteBatch, position: Vector2, layerDepth: float) : void
+ Draw(spriteBatch: SpriteBatch, position: Vector2, layerDepth: float, spriteEffect: SpriteEffects) : void
ExtractCollisionData(texture: Texture2D) : Color[]
+ DeepCopy() : AnimatingSprite

World:

Namespace: ZoneGame

Inherit: /

Implement: /

Descrizione:

Classe che contiene i campi e le definizioni del mondo virtuale.



Component:

Namespace: ZoneGame

Inherit: /

Implement: IComponent

Descrizione: /

<i>Component</i>
<pre># index: int # position: Vector2 # sourceRectangle: Rectangle? # color: Color # origin: Vector2 # scale: float # rotation: float # alphaChannel: float # effects: SpriteEffects # layerDepth: float</pre>
<pre># Component() + Component(index: int) + IsActive { get; set; } : bool + Index { set; } : int + SourceRectangle { get; set; } : Rectangle? + Color { get; set; } : Color + Origin { get; set; } : Vector2 + Scale { get; set; } : float + Rotation { get; set; } : float + AlphaChannel { get; set; } : float + Effects { get; set; } : SpriteEffects + LayerDepth { get; set; } : float + Position { get; set; } : Vector2 + Update(gameTime: GameTime) : void + Draw(spriteBatch: SpriteBatch, gameTime: GameTime) : void + Height() : int + Width() : int</pre>

Selectable

Namespace: ZoneGame

Inherit: /

Implement: ISelectable

Descrizione:

Classe per la gestione di eventi.

Selectable
+ selected: EventHandler
+ Selectable()
+ IsSelectable { get; set; } : bool
+ Selected { get; set; } : EventHandler
+ OnSelected() : void

InGameComponent:

Namespace: ZoneGame

Inherit: /

Implement: ISampleComponent, ICollision

Descrizione: /

InGameComponent

```
# position: Vector2
# origin: Vector2
# color: Color
# scale: float
# rotation: float
# alphaChannel: float
# layerDepth: float
# destinationRectangle: Rectangle
# sourceRectangle: Rectangle?
# effects: SpriteEffects

+ InGameComponent()
+ MatrixTransform { get; } : Matrix
+ LayerDepthRectangle { get; } : Rectangle
+ Bounds { get; } : Rectangle
+ CentralCollisionArea { get; } : Rectangle
+ BottomCollisionArea { get; } : Rectangle
+ OutOfWorldCollisionArea { get; } : Rectangle
+ Position { get; set; } : Vector2
+ Color { get; set; } : Color
+ Origin { get; set; } : Vector2
+ Scale { get; set; } : float
+ Rotation { get; set; } : float
+ AlphaChannel { get; set; } : float
+ DestinationRectangle { get; set; } : Rectangle
+ SourceRectangle { get; set; } : Rectangle?
+ Effects { get; set; } : SpriteEffects
+ Update(gameTime: GameTime) : void
+ Draw(spriteBatch: SpriteBatch, gameTime: GameTime) : void
+ Width() : int
+ Height() : int
```

AlienManager:

Namespace: ZoneGame

Inherit: /

Implement: ISampleComponent

Descrizione:

Classe per la creazione e la gestione delle istanze della classe Alien

AlienManager
<pre> - spawnTimer: TimeSpan - timeChangeDanger: TimeSpan - changeDangerState: TimeSpan - dangerState: Danger - maxNumAlien: int - random: Random # idleSprite: AnimatingSprite # walkingSprite: AnimatingSprite # dyingSprite: AnimatingSprite # attackSprite: AnimatingSprite - isSpawn: bool - score: int - world: World - camera: Camera - aliens: List<Alien> # flockParams: AIParameters # playerActor: Player + AlienManager(idleSprite: AnimatingSprite, walkingSprite: AnimatingSprite, dyingSprite: AnimatingSprite, attackSprite: AnimatingSprite, world: World, camera: Camera, flockParams: AIParameters, playerActor: Player) + DangerState { get; } : Danger + IsSpawn { get; set; } : bool + Score { get; } : int + Aliens { get; } : List<Alien> + FlockParams { get; set; } : AIParameters + Update(gameTime: GameTime) : void + Draw(spriteBatch: SpriteBatch, gameTime: GameTime) : void + ResetFlock() : void + ChangeState(gameTime: GameTime) : void + Spawn(gameTime: GameTime) : void # SetSpawnTimer() : void # SetCardinalDirection() : CardinalDirection # AdjustPositionY(tempAlien: Alien) : void # AdjustPositionX(tempAlien: Alien) : void </pre>

BulletManager:

Namespace: ZoneGame

Inherit: /

Implement: ISampleComponent

Descrizione:

Classe per la creazione e gestione delle istanze della classe Bullet.

BulletManager
<pre> - bulletTexture: Texture2D - bulletTextureData: Color[] - worldSize: Vector2 - bullets: List<Bullet> + BulletManager(bulletTexture: Texture2D, worldSize: Vector2) + Bullets { get; } : List<Bullet> + Update(gameTime: GameTime) : void + Draw(spriteBatch: SpriteBatch, gameTime: GameTime) : void + GenerateBullet(position: Vector2, velocity: Vector2, color: Color) : void # ExtractCollisionData(texture: Texture2D) : Color[] </pre>

EngineManager:

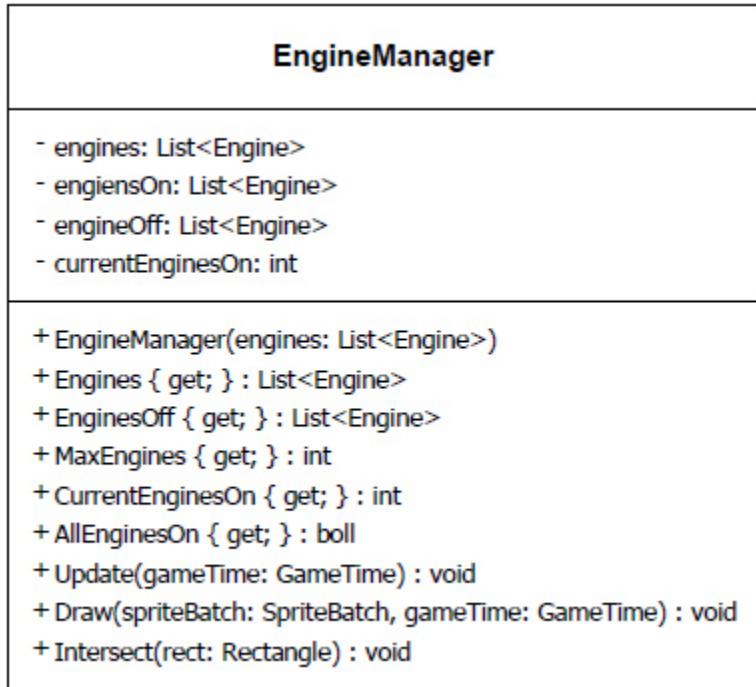
Namespace: ZoneGame

Inherit: /

Implement: ISampleComponent

Descrizione:

Classe per la gestione delle istanze della classe Engine.



ThumbstickComponent:

Namespace: ZoneGame

Inherit: /

Implement: ISampleComponent

Descrizione:

Classe per la rappresentazione dei controller di gioco.

ThumbstickComponent

```
- controlstickBoundary: Texture2D  
- controlStick: Texture2D  
- controlStickBoundaryPosition: Vector2  
- controlStickStartedPosition: Vector2  
- controlStickPosition: Vector2  
  
+ ThumbstickComponent(controlstickBoundary: Texture2D, controlStick: Texture2D)  
+ ControlStickBoundaryPosition { get; set; } : Vector2  
+ ControlStickStartedPosition { get; } : Vector2  
+ ControlStickPosition { get; set; } : Vector2  
+ ControlStickBoundaryDimension { get; } : Rectangle  
+ ControlStickDimension { get; } : Rectangle  
+ Update(gameTime: GameTime) : void  
+ Draw(spriteBatch: SpriteBatch, gameTime: GameTime) : void  
+ ResetControlStickPosition() : void
```

Classi ereditarie di Behavior

HuntBehavior:

Namespace: ZoneGame

Inherit: Behavior

Implement: ISampleComponent

Descrizione: /

HuntBehavior

```
+ HuntBehavior(actor: Actor)  
+ Update(otherActor: Actor, aiParams: AIParameters) : void
```

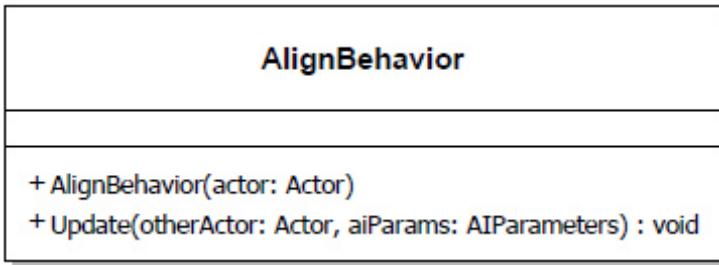
AlignBehavior:

Namespace: ZoneGame

Inherit: Behavior

Implement: /

Descrizione: /



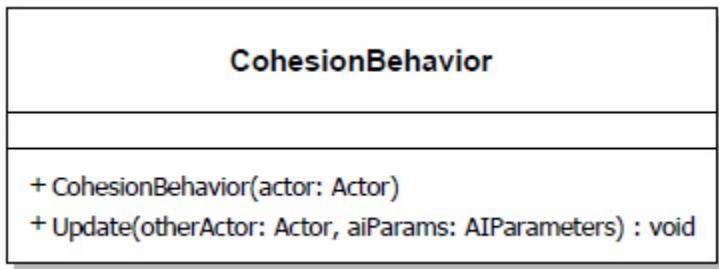
CohesionBehavior:

Namespace: ZoneGame

Inherit: Behavior

Implement: /

Descrizione: /



SeparationBehavior:

Namespace: ZoneGame

Inherit: Behavior

Implement: /

Descrizione: /



Classi ereditarie di Component

Paragraph:

Namespace: ZoneGame

Inherit: Component

Implement: /

Descrizione:

Classe per la creazione un paragrafo di una determinata larghezza.

Paragraph

```
# paragraphBounds: Rectangle
# font: SpriteFont
# text: String

+ Paragraph(font: SpriteFont)
+ Paragraph(font: SpriteFont, bounds: Rectangle, text: String)
+ Paragraph(index: int, font: SpriteFont)
+ Paragraph(index: int, font: SpriteFont, bounds: Rectangle, text: String)
+ Position { get; set; } : Vector2
+ ParagraphBounds { get; set; } : Rectangle
+ Font { get; set; } : SpriteFont
+ Text { get; set; } : String
+ Update(gameTime: GameTime) : void
+ Draw(spriteBatch: SpriteBatch, gameTime: GameTime) : void
+ Width() : int
+ Height() : int
# parseText(Text: String) : String
```

Image:

Namespace: ZoneGame

Inherit: Component

Implement: /

Descrizione:

Classe per la rappresentazione di elementi imagine.

Image

```
- imageContents: Texture2D

+ Image(imageContents: Texture2D)
+ Image(index: int, imageContents: Texture2D)
+ ImageContents { get; set; } : Texture2D
+ Bounds { get; } : Rectangle
+ Update(gameTime: GameTime) : void
+ Draw(spriteBatch: SpriteBatch, gameTime: GameTime) : void
+ Height() : int
+ Width() : int
```

Text:

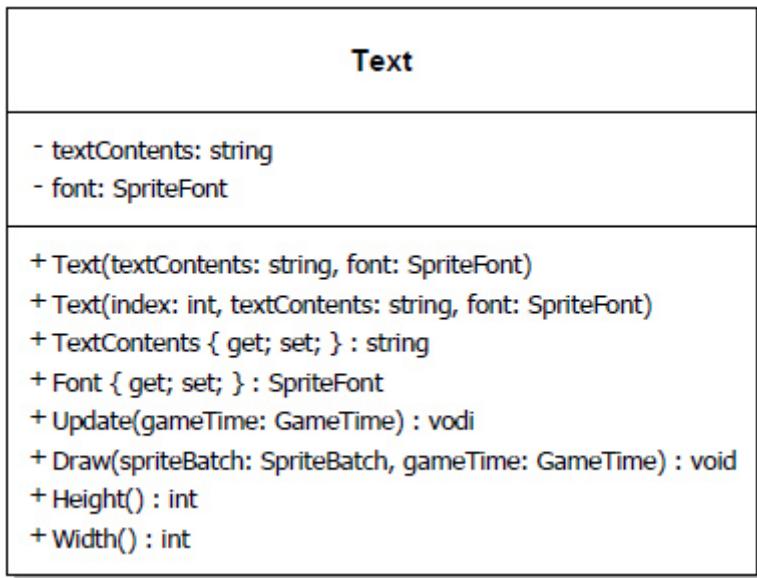
Namespace: ZoneGame

Inherit: Component

Implement: /

Descrizione:

Classe per la rappresentazione di elementi testuali.



TextBox:

Namespace: ZoneGame

Inherit: Component

Implement: /

Descrizione:

Classe per la realizzazione di una textbox di gioco per contenere i dialoghi di gioco.

TextBox

```
- portraitTexture: Texture2D  
- textBoxTexture: Texture2D  
- arrowTexture: Texture2D  
- font: SpriteFont  
- portrait: Image  
- textBox: Image  
- arrow: Image  
- typeWriter: TypeWriterParagraph  
- writerBounds: Rectangle  
- padding: int
```

```
+ TextBox(portraitText: Texture2D, boxText: Texture2D, arrowText: Texture2D, font: SpriteFont)  
+ Position { get; set; } : Vector2  
+ MoveNext { get; set; } : bool  
+ TypeWriterIsIdle { get; } : bool  
+ Bounds { get; } : Rectangle  
+ IsDialogEnded { get; } : bool  
+ Text { get; set; } : String  
+ Update(gameTime: GameTime) : void  
+ Draw(spriteBatch: SpriteBatch, gameTime: GameTime) : void  
+ Width() : int  
+ Height() : int  
+ UpdatePosition() : void
```

SwitchComponent:

Namespace: ZoneGame

Inherit: Component

Implement: /

Descrizione:

Classe per la rappresentazione di elementi a due stati.

SwitchComponent

```
- switchOn: bool  
- imageOn: Texture2D  
- imageOff: Texture2D  
  
+ SwitchComponent(imageOn: Texture2D, imageOff: Texture2D, switchOn: bool)  
+ SwitchComponent(index: int, imageOn: Texture2D, imageOff: Texture2D, switchOn: bool)  
+ SwitchOn { get; set; } : bool  
+ Update(gameTime: GameTime) : void  
+ Draw(spriteBatch: SpriteBatch, gameTime: GameTime) : void  
+ Height() : int  
+ Width() : int
```

Classi ereditarie di Paragraph

TypeWriterParagraph:

Namespace: ZoneGame

Inherit: Paragraph

Implement: /

Descrizione:

Classe per la realizzazione di un paragrafo con testo scritto lettera dopo lettera.

TypeWriterParagraph
<ul style="list-style-type: none"> - isDoneDrawing: bool - isNext: bool - stringToDisplay: List<String> - typedText: String - indexList: int - delayInMilliseconds: int - typedTextLength: double
<ul style="list-style-type: none"> + TypeWriterParagraph(font: SpriteFont) + TypeWriterParagraph(font: SpriteFont, bounds: Rectangle, text: String) + TypeWriterParagraph(index: int, font: SpriteFont) + TypeWriterParagraph(index: int, font: SpriteFont, bounds: Rectangle, text: String) + IsDoneDrawing { get; set; } : bool + IsNext { get; set; } : bool + TypeWriterIsIdle { get; set; } : bool + ParagraphBounds { get; set; } : Rectangle + Font { get; set; } : SpriteFont + Text { get; set; } : string + Update(gameTime: GameTime) : void + Draw(spriteBatch: SpriteBatch, gameTime: GameTime) : void # Reset(text: String) : void # SplitParsedText() : List<String> + ResetSplitParsedText() : void + ResetFields() : void # RestartParagraph() : void # InitializeNext() : void

Classi ereditarie di Image

ImageSelectable:

Namespace: ZoneGame

Inherit: Image

Implement: /

Descrizione:

Classe per la realizzazione di immagini che rispondono all'input dell'utente.

ImageSelectable
- selectable: Selectable
+ ImageSelectable(imageContents: Texture2D)
+ ImageSelectable(index: int, imageContents: Texture2D)
+ IsSelectable { get; set; } : bool
+ Selected { get; set; } : EventHandler
+ OnSelected() : void

Classi ereditarie di ImageSelectable

LevelButton:

Namespace: ZoneGame

Inherit: ImageSelectable

Implement: /

Descrizione:

Classe per la realizzazione dei pulsanti per selezionare un livello nella schermata di scelta del livello.

LevelButton
- font: SpriteFont
- info: InfoLevel
- isUnlocked: bool
- levelNum: int
+ LevelButton(text: Texture2D, font: SpriteFont, info: InfoLevel, levelNum: int)
+ LevelButton(index: int, text: Texture2D, font: SpriteFont, info: InfoLevel, levelNum: int)
+ Info { get; set; } : InfoLevel
+ IsUnlocked { get; set; } : bool
+ LevelNum { get; set; } : int
+ Draw(spriteBatch: SpriteBatch, gameTime: GameTime) : void

Classi ereditarie di Text

IntermittenceText:

Namespace: ZoneGame

Inherit: Text

Implement: /

Descrizione:
Classe per la realizzazione di testo lampeggiante.

IntermittenceText	
- <code>isDoneDrawing: bool</code>	
- <code>isNext: bool</code>	
- <code>stringToDisplay: List<string></code>	
- <code>typedText: String</code>	
- <code>indexList: int</code>	
- <code>delayInMilliseconds: int</code>	
- <code>typedTextLength: double</code>	
+ <code>IntermittenceText(font: SpriteFont)</code>	
+ <code>IntermittenceText(font: SpriteFont, bounds: Rectangle, text: String)</code>	
+ <code>IntermittenceText(index: int, font: SpriteFont)</code>	
+ <code>IntermittenceText(index: int, font: SpriteFont, bounds: Rectangle, text: String)</code>	
+ <code>IsDoneDrawing { get; set; } : bool</code>	
+ <code>IsNext { get; set; } : bool</code>	
+ <code>TypeWriterIsIdle { get; set; } : bool</code>	
+ <code>ParagraphBounds { get; set; } : Rectangle</code>	
+ <code>Font { get; set; } : SpriteFont</code>	
+ <code>Text { get; set; } : string</code>	
+ <code>Update(gameTime: GameTime) : void</code>	
+ <code>Draw(spriteBatch: SpriteBatch, gameTime: GameTime) : void</code>	
# <code>Reset(text: string) : void</code>	
# <code>SplitParsedText() : List<string></code>	
+ <code>ResetSplitParsedText() : void</code>	
+ <code>ResetFields() : void</code>	
# <code>RestartParagraph() : void</code>	
# <code>InitializeNext() : void</code>	

TextSelectable:

Namespace: ZoneGame

Inherit: Text

Implement: /

Descrizione:
Classe per la realizzazione di testo selezionabile.

TextSelectable

```
# selectable: Selectable

+ TextSelectable(textContentes: string, font: SpriteFont)
+ TextSelectable(index: int, textContents: string, font: SpriteFont)
+ IsSelectable { get; set; } : bool
+ Selected { get; set; } : EventHandler
+ OnSelected() : void
```

Classi ereditarie di TextSelectable

TextButton:

Namespace: ZoneGame

Inherit: TextSelectable

Implement: /

Descrizione:

Classe per la realizzazione di un bottone con testo selezionabile.

TextButton

```
- buttonTexture: Texture2D
- alphaTexture: float
- alphaText: float

+ TextButton(textContents: string, font: SpriteFont, buttonTexture: Texture2D)
+ TextButton(index: int, textContents: string, font: SpriteFont, buttonTexture: Texture2D)
+ ButtonTexture { get; set; } : Texture2D
+ AlphaTexture { get; set; } : float
+ AlphaText { get; set; } : float
+ AlphaChannel { get; set; } : float
+ Update(gameTime: GameTime) : void
+ Draw(spriteBatch: SpriteBatch, gameTime: GameTime) : void
+ Height() : int
+ Width() : int
- getTextPosition() : Vector2
```

Classi ereditarie di InGameComponent

InanimateGameComponent:

Namespace: ZoneGame

Inherit: InGameComponent

Implement: /

Descrizione:

Classe di base per tutti gli elementi presenti nella mappa di gioco che non sono animati e sono interessati da collisione.

InanimateGameComponent

```
# idleComponentTexture: Texture2D
# textureData: Color[]
# worldSize: Vector2

+ InanimateGameComponent(idleComponentTexture: Texture2D, worldSize: Vector2)
+ TextureData { get; set; } : Color[]
+ MatrixTransform { get; } : Matrix
+ IdleComponentTexture { get; } : Texture2D
+ BoundsTransform { get; } : Rectangle
+ Bounds { get; } : Rectangle
+ CentralCollisionArea { get; } : Rectangle
+ BottomCollisionArea { get; } : Rectangle
+ WorldSize { get; set; } : Vector2
+ Draw(spriteBatch: SpriteBatch, gameTime: GameTime) : void
+ Width() : int
+ Height() : int
# CalculateBoundingRectangle(rectangle: Rectangle, transform: Matrix) : Rectangle
# CalcolateLayerDepth() : float
```

Actor:

Namespace: ZoneGame

Inherit: InGameComponent

Implement: /

Descrizione:

Classe base per tutti i componenti della mappa che influenzano o sono influenzati da altri componenti della mappa di tipo Actor.

Actor
<pre># moveSpeed: float # behaviors: Dictionary<ActorType, Behaviors> # actorType: ActorType # reactionDistance: float # worldSize: Vector2 # direction: Vector2 + Actor(worldSize: Vector2) + ActorType { get; } : ActorType + ReactionDistance { get; } : float + ReactionLocation { get; } : Vector2 + WorldSize { get; set; } : Vector2 + Direction { get; } : Vector2</pre>

Arrow:

Namespace: ZoneGame

Inherit: InGameComponent

Implement: /

Descrizione: /

Arrow
<pre>- hide: bool - arrowTexture: Texture2D - distanceRadius: float - objective: InGameComponent + Arrow(text: Texture2D, obj: InGameComponent) + DistanceRadius { get; set; } : float + Objective { get; } : InGameComponent + Update(gameTime: GameTime) : void + Draw(spriteBatch: SpriteBatch, gameTime: GameTime) : void # CalcolateArrowsRotation() : void</pre>

Classi ereditarie di InanimateGameComponent

Grass:

Namespace: ZoneGame

Inherit: InanimateGameComponent

Implement: /

Descrizione: /

Grass
+ Grass(text: Texture2D, worldSize: Vector2) + LayerDepthRectangle { get; } : Rectangle + BottomCollisionArea { get; } : Rectangle + Draw(spriteBatch: SpriteBatch, gameTime: GameTime) : void

Tree:

Namespace: ZoneGame

Inherit: InanimateGameComponent

Implement: /

Descrizione: /

Tree
+ Tree(idleComponentTexture: Texture2D, worldSize: Vector2) + LayerDepthRectangle { get; } : Rectangle + Bounds { get; } : Rectangle + CentralCollisionArea { get; } : Rectangle + BottomCollisionArea { get; } : Rectangle + Draw(spriteBatch: SpriteBatch, gameTime: GameTime) : void

Bush:

Namespace: ZoneGame

Inherit: InanimateGameComponent

Implement: /

Descrizione: /

Bush

```
+ Bush(text: Texture2D, worldSize: Vector2)
+ LayerDepthRectangle { get; } : Rectangle
+ BottomCollisionArea { get; } : Rectangle
```

SpaceShip:

Namespace: ZoneGame

Inherit: InanimateGameComponent

Implement: /

Descrizione: /

SpaceShip

```
+ SpaceShip(idleComponentTexture: Texture2D, worldSize: Vector2)
+ LayerDepthRectangle { get; } : Rectangle
+ Bounds { get; } : Rectangle
+ CentralCollisionArea { get; } : Rectangle
+ Draw(spriteBatch: SpriteBatch, gameTime: GameTime) : void
```

ScoreBarF:

Namespace: ZoneGame

Inherit: InanimateGameComponent

Implement: /

Descrizione:

Classe per la realizzazione di una barra, che utilizza valori a virgola mobile.

ScoreBarF
<pre> - scoreBarOrientation: ScoreBarOrientation - height: int - width: int - currentValue: float - backgroundTexture: Texture2D - redTexture: Texture2D - greenTexture: Texture2D - yellowTexture: Texture2D + ScoreBarF(backgroundTexture: Texture2D, greenTexture: Texture2D, yellowTexture: Texture2D, redTexture: Texture2D, minValue: float, maxValue: float, height: int, width: int, scoreBarColor: Color, scoreBarOrientation: ScoreBarOrientation, initialValue: int) + MinValue { get; set; } : float + MaxValue { get; set; } : float + ScoreBarColor { get; set; } : Color + CurrentValue { get; } : float + IsMaxCurrentValue { get; } : bool + Draw(spriteBatch: SpriteBatch, gameTime: GameTime) : void + IncreaseCurrentValue(valueToIncrease: float) : void + DecreaseCurrentValue(valueToDecrease: float) : void + DecreaseCurrentValue(valueToDecrease: float, clampToMinimum: bool) : int + SetToMaxValue() : void + SetToMinValue() : void - GetSpaceFromBorder() : decimal - GetTextureByCurrentValue(value: float) : Texture2D + Width() : int + Height() : int </pre>

ScoreBar:

Namespace: ZoneGame

Inherit: InanimateGameComponent

Implement: /

Descrizione:

Classe per la realizzazione di una barra, che utilizza valori interi.

ScoreBar
<pre> - scoreBarOrientation: ScoreBarOrientation - height: int - width: int - currentValue: int - backgroundTexture: Texture2D - redTexture: Texture2D - greenTexture: Texture2D - yellowTexture: Texture2D + ScoreBar(backgroundTexture: Texture2D, greenTexture: Texture2D, yellowTexture: Texture2D, redTexture: Texture2D, minValue: int, maxValue: int, height: int, width: int, scoreBarColor: Color, scoreBarOrientation: ScoreBarOrientation, initialValue: int) + MinValue { get; set; } : int + MaxValue { get; set; } : int + ScoreBarColor { get; set; } : Color + CurrentValue { get; } : int + IsMaxCurrentValue { get; } : bool + Draw(spriteBatch: SpriteBatch, gameTime: GameTime) : void + IncreaseCurrentValue(valueToIncrease: int) : void + DecreaseCurrentValue(valueToDecrease: int) : void + DecreaseCurrentValue(valueToDecrease: int, clampToMinimum: bool) : int - GetSpaceFromBorder() : decimal - GetTextureByCurrentValue(value: int) : Texture2D + Width() : int + Height() : int </pre>

ObjectiveFinder:

Namespace: ZoneGame

Inherit: InanimateGameComponent

Implement: /

Descrizione:

Classi per il mantenimento e la gestione degli obiettivi di gioco.

ObjectiveFinder

```
- baseTexture: Texture2D  
- arrowTexture: Texture2D  
- center: Vector2  
- arrows: List<Arrow>  
- arrowOrigin: Vector2  
- distanceRadius: float  
  
+ ObjectiveFinder(baseTexture: Texture2D, arrowTexture: Texture2D)  
+ Center { get; } : Vectro2  
+ CentralPosition { get; } : Vector2  
+ Arrows { get; set; } : List<Arrow>  
+ Update(gameTime: GameTime) : void  
+ Draw(spriteBatch: SpriteBatch, gameTime: GameTime) : void  
# Distance(point1: Vector2, point2: Vector2) : bool  
+ AddObjectives(obj: InGameComponent) : void  
+ RemoveObjectives(obj: InGameComponent) : void  
+ Width() : int  
+ Height() : int
```

BulletReloadBar:

Namespace: ZoneGame

Inherit: InanimateGameComponent

Implement: /

Descrizione: /

BulletReloadBar

```
- player: Player  
- bar: ScoreBarF  
- reset: bool  
- dTime: TimeSpan  
  
+ BulletReloadBar(player: Player, barBorderText: Texture2D, barInnerText: Texture2D)  
+ Position { get; set; } : Vector2  
+ Update(gameTime: GameTime) : void  
+ Draw(spriteBatch: SpriteBatch, gameTime: GameTime) : void  
+ Width() : int  
+ Height() : int  
- UpdateBarState() : void
```

Bullet:

Namespace: ZoneGame

Inherit: InanimateGameComponent

Implement: /

Descrizione: /

Bullet
- velocity: Vector2
+ Bullet(idleComponentTexture: Texture2D, worldSize: Vector2, pos: Vector2, vel: Vector2, col: Color)
+ LayerDepthRectangle { get; } : Rectangle
+ Update(gameTime: GameTime) : void
+ Draw(spriteBatch: SpriteBatch, gameTime: GameTime) : void
+ PlayBulletCollision() : void

Engine:

Namespace: ZoneGame

Inherit: InanimateGameComponent

Implement: /

Descrizione: /

Engine

```
# engineOnTexture: Texture2D
# engineOffTexture: Texture2D
# activeTexture: Texture2D
# scoreBar: ScoreBar
# on: bool

+ Engine(engineOnTexture: Texture2D, engineOffTexture: Texture2D, scoreBar: ScoreBar, on: bool)
+ Position { get; set; } : Vector2
+ Bounds { get; } : Rectangle
+ CentralCollisionArea { get; } : Rectangle
+ On { get; set; } : bool
+ IsEnergyFull { get; } : bool
+ Update(gameTime: GameTime) : void
+ Draw(spriteBatch: SpriteBatch, gameTime: GameTime) : void
+ Width() : int
+ Height() : int
+ IncreaseEnergy(amount: int) : void
```

Classi ereditarie di Actor

Standard Character:

Namespace: ZoneGame

Inherit: Actor

Implement: /

Descrizione:

Classe base per la realizzazione di una personaggio standard.

StandardCharacter

```
# moveAmount: Vector2
# strSounds: string[]
# isMoving: bool
# hadDirectionChanged: bool
# cardinalDirectionOffset: float
- state: CharacterState
# isHit: bool
- cardDirection: CardinalDirection
- currentAnimatingSprite: AnimatingSprite
- idleSprite: AnimatingSprite
- walkingSprite: AnimatingSprite
- dyingSprite: AnimatingSprite

+ StandardCharacter(idleSprite: AnimatingSprite, walkingSprite: AnimatingSprite, dyingSprite: AnimatingSprite, worldSize: Vector2)
+ IsMoving { get; } : bool
+ HadDirectionChanged { get; } : bool
+ MoveAmount { get; } : Vector2
+ BodySize { get; } : Vector2
+ HitPosition { get; set; } : Vector2
+ HitPoint { get; } : Vector2
+ BodyBounds { get; } : Rectangle
+ FrameBounds { get; } : Rectangle
+ LayerDepthRectangle { get; } : Rectangle
+ Bounds { get; } : Rectangle
+ CentralCollisionArea { get; } : Rectangle
+ BottomCollisionArea { get; } : Rectangle
+ OutOfWorldCollisionArea { get; } : Rectangle
+ State { get; set; } : CharacterState
+ IsDeadOrDying { get; } : bool
+ IsDead { get; } : bool
+ IsHit { get; set; } : bool
+ CardDirection { get; set; } : CardinalDirection
+ CurrentFrameSize { get; } : Vector2
+ CurrentFrameDimension { get; } : Rectangle
+ CurrentFrameCenter { get; } : Vector2
+ CurrentAnimatingSprite { get; set; } : AnimatingSprite
+ IdleSprite { get; set; } : AnimatingSprite
+ WalkingSprite { get; set; } : AnimatingSprite
+ DyingSprite { get; set; } : AnimatingSprite
+ Update(gameTime: GameTime) : void
+ Draw(spriteBatch: SpriteBatch, gameTime: GameTime) : void
# UpdateCharacterState() : void
# UpdateDirectionChanged() : void
+ Move(movement: Vector2, move: bool, gameTime: GameTime) : void
# SetMovement(movement: Vector2, gameTime: GameTime) : void
# CalcolateCardinalDirection(vector: Vector2) : CardinalDirection
- DeterminateCardinalDirectionDomainX(vector: Vector2, offset: float) : CardinalDirection
- DeterminateCardinalDirectionDomainY(vector: Vector2, offset: float) : CardinalDirection
+ Width() : int
+ Height() : int
+ ResetAnimation(isWalking: bool) : void
+ AnimateStateIdle() : void
+ AnimateStateWalking() : void
+ AnimateStateDead() : void
+ DirectionChanged() : void
+ UpdateAnimation(gameTime: GameTime) : void
# CalcolateLayerDepth() : float
# PlaySound(strSound: string) : void
# PlaySound(strSound: string, loop: bool, volume: float) : void
```

Cassie ereditarie di StandardCharacter

MonsterCharacter:

Namespace: ZoneGame

Inherit: StandardCharacter

Implement: /

Descrizione: /

MonsterCharacter
isAttacking: bool - attackSprite: AnimatingSprite
+ MonsterCharacter(idleSprite: AnimatingSprite, walkingSprite: AnimatingSprite, dyingSprite: AnimatingSprite, attackSprite: AnimatingSprite, worldSize: Vector2) + IsAttacking { get; } : bool + AttackSprite { get; set; } : AnimatingSprite # UpdateCharacterState() : void # UpdateDirectionChanged() : void + ResetAnimation(isWalking: bool) : void + AnimateStateAttack() : void + DirectionChanged() : void

RiflemanCharacter:

Namespace: ZoneGame

Inherit: StandardCharacter

Implement: /

Descrizione: /

RiflemanCharacter
<pre># hadAimDirectionChanged: bool # isAiming: bool # isRotating: bool - aimCardDirection: CardinalDirection - aimSprite: AnimatingSprite + RiflemanCharacter(idleSprite: AnimatingSprite, walkingSprite: AnimatingSprite, dyingSprite: AnimatingSprite, aimSprite: AnimatingSprite, worldSize: Vector2) + HadAimDirectionChanged { get; } : bool + IsAiming { get; } : bool + IsRotating { get; set; } : bool + AimCardDirection { get; set; } : CardinalDirection + AimSprite { get; set; } : AnimatingSprite + Draw(spriteBatch: SpriteBatch, gameTime: GameTime) : void # UpdateCharacterState() : void # UpdateDirectionChanged() : void + SetRotation(movement: Vector2) : void - GetSpriteEffect(movementDirection: Vector2) : void - CalculateAimCardDirection(movement: Vector2) : CardinalDirection - AngleVector(angle: float) : Vector2 - CalculateCardDirectionFromAimDirection(aimDir: CardinalDirection) : CardinalDirection + ResetAnimation(isWalking: bool) : void + AnimateStateAiming() : void + AimDirectionChanged() : void</pre>

Classi ereditarie di MonsterCharacter

Alien:

Namespace: ZoneGame

Inherit: MonsterCharacter

Implement: /

Descrizione: /

Alien
<pre> - attackRadius: float - random: Random - aiNewDir: Vector2 - aiNumSeen: int - boundryWidth: int - boundryHeight: int - playerActor: Player + preyReached: bool - removeAlien: bool - timeToRemove: TimeSpan - attackTimer: TimeSpan - attackTimeElapsed: TimeSpan + Alien(idleSprite: AnimatingSprite, walkingSprite: AnimatingSprite, dyingSprite: AnimatingSprite, attackSprite: AnimatingSprite, worldSize: Vector2, playerActor: Player) + BodyBounds { get; } : Rectangle + HitPoint { get; } : Vector2 + IsIndisposed { get; } : bool + RemoveAlien { get; } : bool + Update(gameTime: GameTime, inout aiParams: AIParameters) : void + Draw(spriteBatch: SpriteBatch, gameTime: GameTime) : void + BuildBehaviors() : void + ResetThink() : void - ClosestLocation(inout srcLocation: Vector2) : void + ReactTo(actor: Actor, inout AIParams: AIParameters) : void + ContainsPoint(point1: Vector2, point2: Vector2, distance: float) : bool + IsPlayerHit() : bool - CalculateAngle(inout distance: float, inout angle: float, point: Vector2) : void # CalculateLayerDepth() : float + HandleHitState() : void </pre>

Classi ereditarie di RiflemanCharacter

Player:

Namespace: ZoneGame

Inherit: RiflemanCharacter

Implement: /

Descrizione: /

```

Player

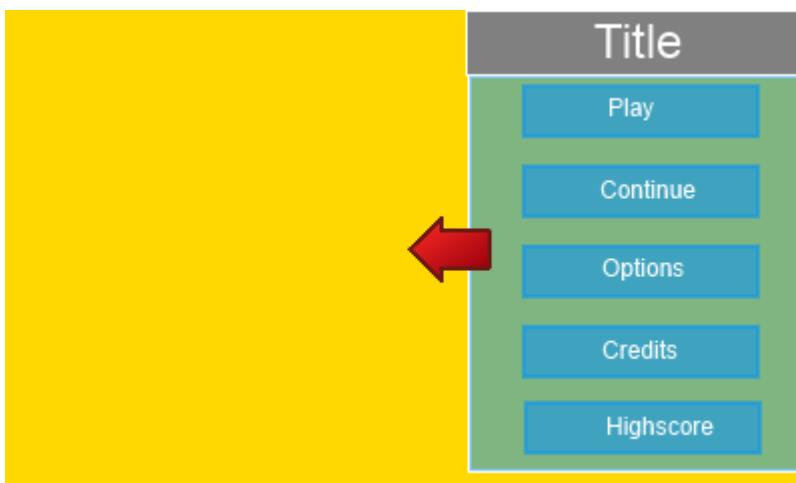
# isSoundDeadActive: bool
# aimAngle: float
- aimLine: Texture2D
- aimLineCenter: Vector2
- aimLineVelocity: Vector2
- currentEffect: SpriteEffects
- aimLineRotation: float
# bulletMang: BulletManager
- bulletSpeed: float
- cooldown: TimeSpan
- fireTimer: TimeSpan
- bodySize: Vector2

+ Player(idleSprite: AnimatingSprite, walkingSprite: AnimatingSprite, dyingSprite: AnimatingSprite, aimSprite: AnimatingSprite, worldSize: Vector2, aimLine: Texture2D, bulletTexture: Texture2D)
+ AimAngle { get; } : float
+ AimLineVelocity { get; set; } : Vector2
- AimLinePosition { get; } : Vector2
+ AimLineRotation { get; } : float
+ ThumstickArea { get; set; } : Rectangle
+ BulletMang { get; } : BulletManager
+ CoolDown { get; } : TimeSpan
+ FireTimer { get; } : TimeSpan
+ HitPoint { get; } : Vector2
+ basePosition { get; } : Vector2
+ LayerDepthRectangle { get; } : Rectangle
+ Bounds { get; } : Rectangle
+ CentralCollisionArea { get; } : Rectangle
+ BottomCollisionArea { get; } : Rectangle
+ GroundCollisionArea { get; } : Rectangle
+ OutOfWorldCollisionArea { get; } : Rectangle
+ Update(gameTime: GameTime) : void
+ Draw(spriteBatch: SpriteBatch, gameTime: GameTime) : void
# SetMovement(movement: Vector2, gameTime: GameTime) : void
+ CalcolateDirection(movement: Vector2) : Vector2
+ SetRotation(movement: Vector2) : void
+ GenerateBullet() : void

```

3.3 Prototipi Interfaccia Grafica

Main Menu



All'avvio dell'applicazione viene visualizzata la seguente schermata.

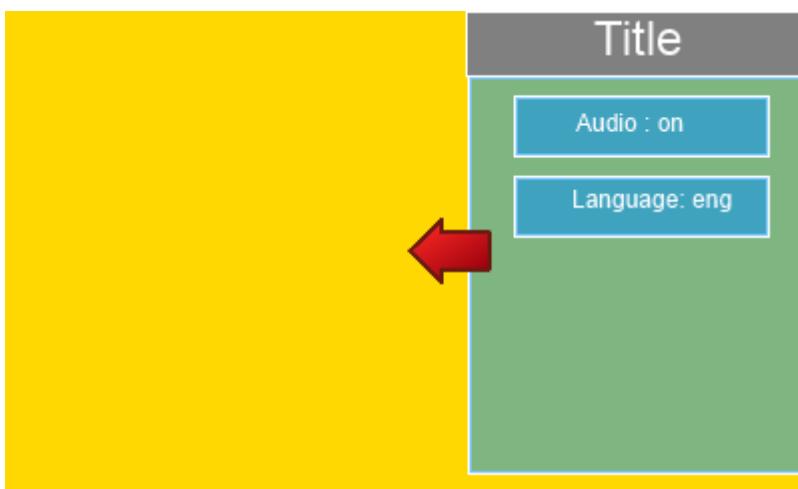
Descrizione:

La finestra contenitore esce lateralmente dal lato destro dello schermo. Raggiunta la sua posizione finale interrompe lo spostamento orizzontale.

Gli elementi con cui l'utente può interagire sono:

- Play → Visualizza la schermata Livelli per la scelta del livello
- Continue → Avvia il gioco (questo elemento è presente solo se esiste un salvataggio della partita in memoria)
- Options → Visualizza la schermata Opzioni
- Credits → Visualizza la schermata Crediti
- Highscore → Visualizza la schermata Lista Punteggi

Opzioni



Viene visualizzata dopo che l'utente ha selezionato il riquadro Opzioni del Main Menu.

Descrizione:

La finestra contenitore esce lateralmente dal lato destro dello schermo. Raggiunta la sua posizione finale interrompe lo spostamento orizzontale.

Gli elementi selezionabili dall'utente sono:

- Audio → Abilita/Disabilita gli effetti sonori. La selezione comporta la modifica del testo tra due valori On e Off.
- Linguaggio → Modifica il linguaggio in Italiano/Inglese. La selezione comporta la modifica del testo tra due valori Ita e Eng.

Lista Punteggi

Place	Player Name	Score
string	string	string

Viene visualizzata quando l'utente seleziona il riquadro Punteggi del MainMenu.

Descrizione:

La barra superiore e inferiore all'avvio della schermata sono chiuse e si aprono mostrando la schermata.

Gli elementi selezionabili sono:

- Previous (indicato dalla freccia rivolta verso sinistra) → Visualizza la pagina precedente se esiste.
- Next(indicato dalla freccia rivolta verso sinistra) → Visualizza la pagina successiva se esiste.

Crediti

Actor	name surname	name surname
Actor	name surname	name surname
Actor	name surname	name surname
Actor	name surname	name surname

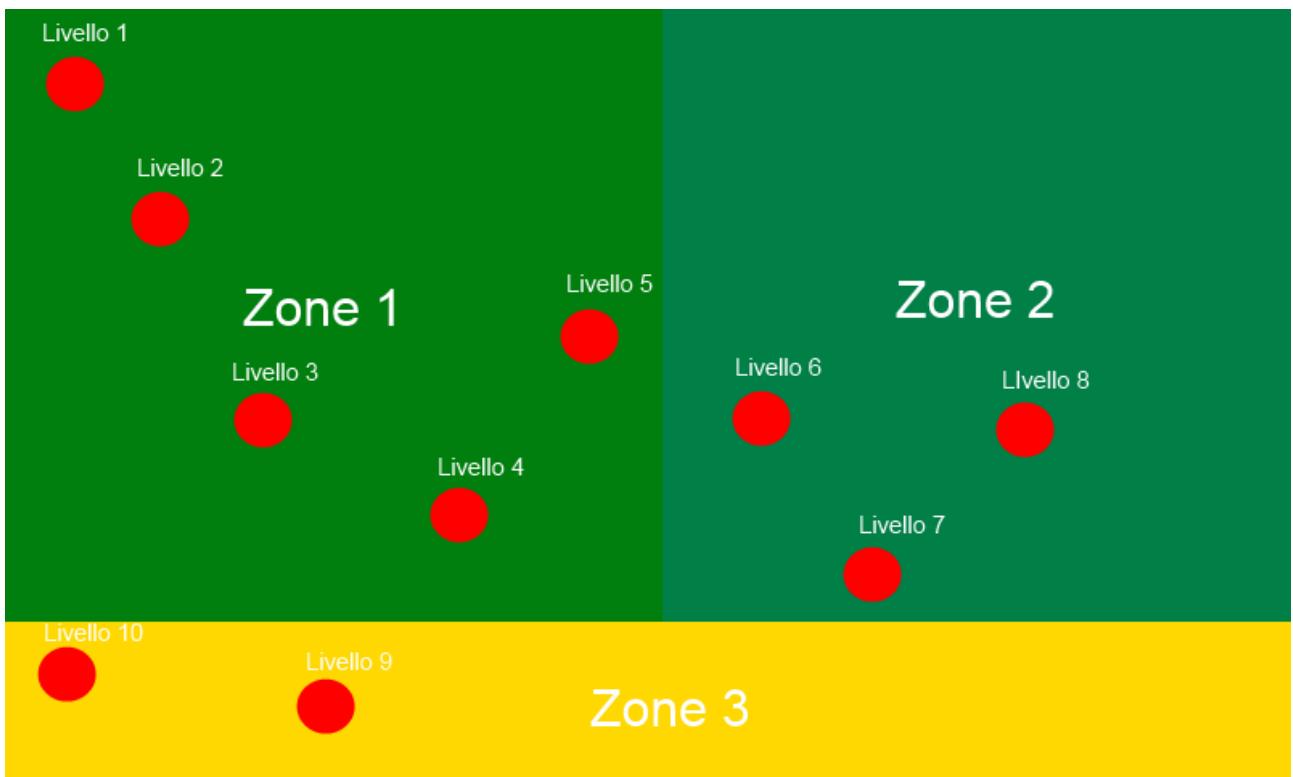
Viene visualizzata quando l'utente seleziona il riquadro Crediti del MainMenu.

Descrizione:

Il testo parte dal basso e scorre verso l'alto.

Non ci sono elementi selezionabili.

Schermata Livelli

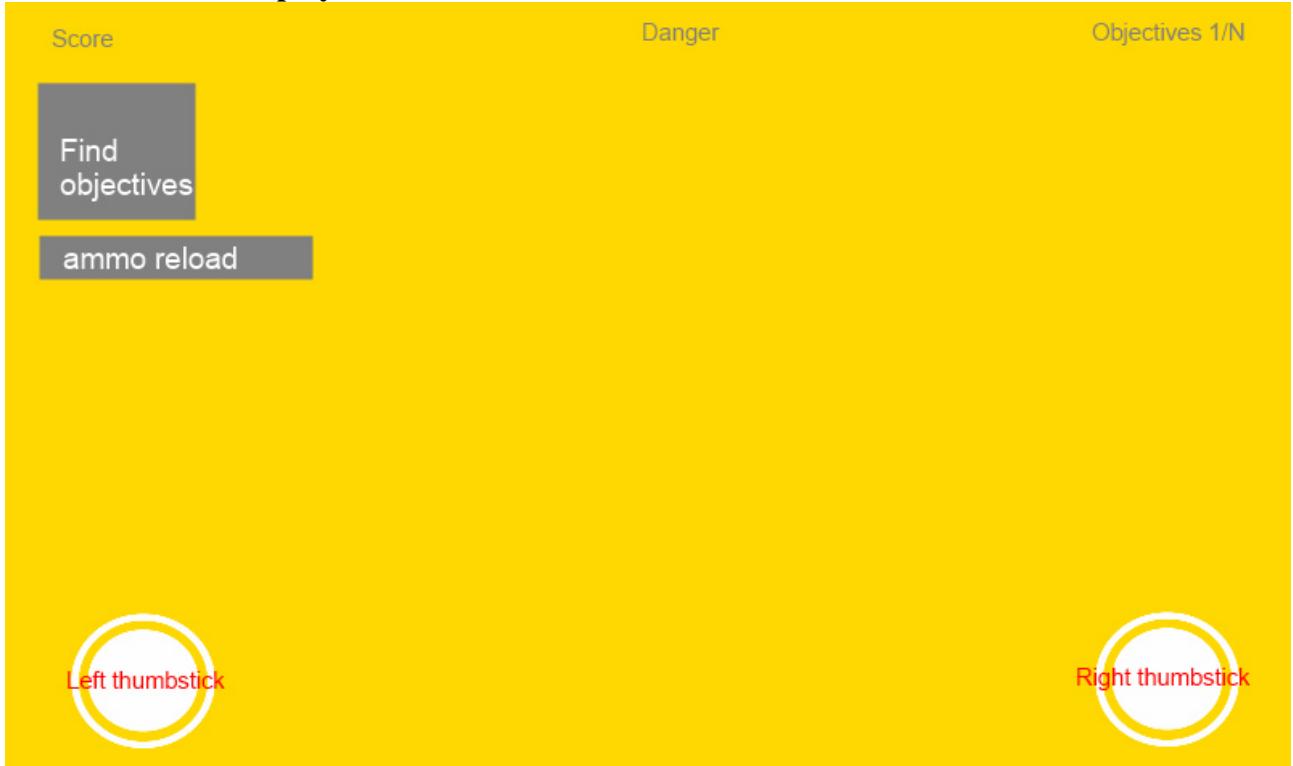


Viene visualizzata quando l'utente avvia una nuova partita.

Gli elementi selezionabili sono:

- Oggetti circolari → Avvia il livello selezionato.

Schermata di Gameplay

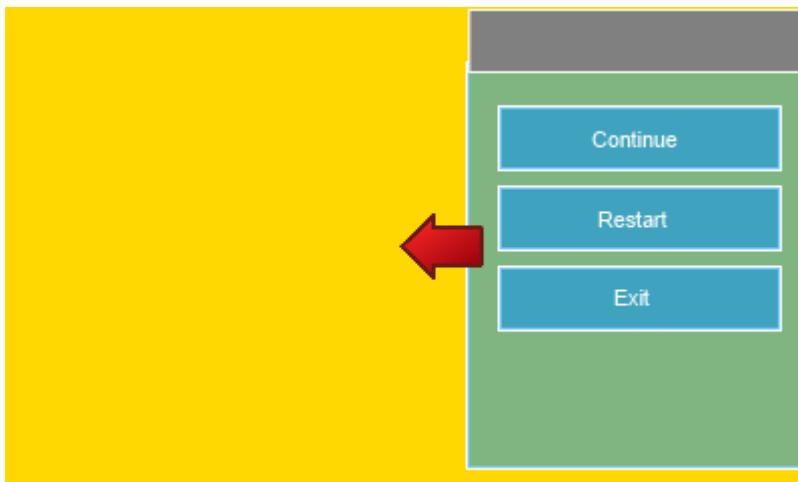


Rappresenta la schermata di gioco.

Gli elementi selezionabili sono:

- Left thumbstick → Controlla il movimento del giocatore.
- Right thumbstick → Permette di prendere la mira.
- Find objectives → Visualizza la direzione degli obiettivi da completare.

Menu di Pausa



Rappresenta il menu di pausa.

Descrizione:

La finestra contenitore esce lateralmente dal lato destro dello schermo verso sinistra. Raggiunta la sua posizione finale interrompe lo spostamento orizzontale.

Gli elementi selezionabili dall'utente sono:

- Continue → Continua la partita.
- Restart → Ricomincia il livello.
- Exit → Termina la partita e torna al menu principale.