Parasitic

Description

Parasitic is a top down, turn based strategy game that allows players to explore to become strong enough to fight back a parasitic outbreak. Each level player units are level 1 and will need to explore and kill small enemies to upgrade to take on the bigger threats in the area.

Prerequisites

Installation

Contributing

Issue tracker: https://github.com/Reknotx/Parasitic/issues

License

Citation

Contact

Email addresses

- chaseoconnor3@comcast.net
- croconnor@mail.csuchico.edu

Table of Contents

Table of Contents

PAGE BREAK!

Class Combat System

Fields

AbilityInfo

Declaration

```
public Image abilityInfo;
```

Field Value

Type	Description
Image	The image detailing the info for the ability.

AbilityOneCDText

Declaration

```
public Text abilityOneCDText;
```

Field Value

Type	Description
Text	Displays the remaining cooldown on ability one.

AbilityTwoCDText

Declaration

```
public Text abilityTwoCDText;
```

Field Value

Type	Description
Text	Displays the remaining cooldown on ability two.

ActiveSideImage

Declaration

```
public Image activeSideImage;
```

Field Value

Type	Description
Image	Image showing icons of which side is active in the game.

ActiveSideTextImage

Declaration

```
public Image activeSideTextImage;
```

Field Value

Type	Description
Image	Image of the text stating which side is active in the game.

ActiveUnits

Declaration

```
public ActiveUnits activeUnits;
```

Field Value

Type	Description
ActiveUnits	Indicates which units are currently active.

ArcherHealthSlider

Declaration

```
public Slider archerHealthSlider;
```

Field Value

Type	Description
Slider	The slider UI component representing the Archer's health in graphical form.

ArcherHealthText

Declaration

```
public Text archerHealthText;
```

Field Value

Type	Description
Text	The text UI component representing the Archer’s health in numerical form.

ArcherIcon

Declaration

```
public Image archerIcon;
```

Field Value

Type	Description
Image	The image UI component that contains the Archer’s UI icon.

Blood

Declaration

```
public ParticleSystem blood;
```

Field Value

Type	Description
Particle System	The bloody particle system to be played when a player unit has died.

BloodAndGuts

Declaration

```
public ParticleSystem bloodAndGuts;
```

Field Value

Type	Description
Particle System	The bloody particle system to be played when an enemy unit has died.

CombatButtons

Declaration

```
public List<Button> combatButtons = new List<Button>( );
```

Field Value

Type	Description
List<Button>	The list containing all the buttons used for combat.

CoolingTiles

Declaration

```
public List<Tile> coolingTiles = new List<Tile>( );
```

Field Value

Type	Description
List<Tile>	TODO. Ask Ryan

DefendInfoSprite

Declaration

```
public Sprite defendInfoSprite;
```

Field Value

Type	Description
Sprite	TODO

EndCanvas

Declaration

```
public GameObject endCanvas;
```

Field Value

Type	Description
Game Object	The canvas object that is displayed when the level has been won or lost.

EndGameText

Declaration

```
public Text endGameText;
```

Field Value

Type	Description
Text	The text component that tells the player if they have won, or lost, the level.

EnemiesAliveText

Declaration

```
public Text enemiesAliveText;
```

Field Value

Type	Description
Text	Text object indicating the number of enemies remaining alive.

EnemiesToGo

Declaration

```
private List<Enemy> enemiesToGo = new List<Enemy>( );
```

Field Value

Type	Description

List<Enemy>	Private list that is populated with all of the active and revealed enemy objects in the level.

EnemyTurnSprite

Declaration

```
public Sprite enemyTurnSprite;
```

Field Value

Type	Description
Sprite	The sprite used to indicate it is the enemy’s turn. Contains the enemy icons.

EnemyTurnTextSprite

Declaration

```
public Sprite enemyTurnTextSprite;
```

Field Value

Type	Description
Sprite	The sprite that contains text indicating it is the enemy’s turn.

IgnoreDoubleMoveCheck

Declaration

```
public bool IgnoreDoubleMoveCheck = false;
```

Field Value

Type	Description
bool	Used to prevent player units from receiving a boost in their move speed when no enemies are currently revealed in the level.

Remarks

This is false by default.

This is only needed on levels that are too small to warrant a move speed boost. Currently it is only used in, and needed for, the tutorial level as all other levels are massive in scale.

Instance

Declaration

```
public static CombatSystem Instance;
```

Field Value

Type	Description
Combat System	A reference to the singleton instance of the Combat System script in the scene.

KnightHealthSlider

Declaration

```
public Slider knightHealthSlider;
```

Field Value

Type	Description
Slider	The slider UI component representing the Knight’s health in graphical form.

KnightHealthText

Declaration

```
public Text knightHealthText;
```

Field Value

Type	Description
Text	The text UI component representing the Knight’s health in numerical form.

KnightIcon

Declaration

```
public Image knightIcon;
```

Field Value

Type	Description
Image	The image UI component that contains the Knight’s UI icon.

MageHealthSlider

Declaration

```
public Slider mageHealthSlider;
```

Field Value

Type	Description
Slider	The slider UI component representing the Mage’s health in graphical form.

MageHealthText

Declaration

```
public Text mageHealthText;
```

Field Value

Type	Description
Text	The text UI component representing the Mage’s health in numerical form.

MageIcon

Declaration

```
public Image mageIcon;
```

Field Value

Type	Description

Image	The image UI component that contains the Mage’s UI icon.

PlayersToGo

Declaration

```
private List<Player> playersToGo = new List<Player>( );
```

Field Value

Type	Description
List<Player>	Private list populated with all the player units

PlayerTurnSprite

Declaration

```
public Sprite playerTurnSprite;
```

Field Value

Type	Description
Sprite	The sprite image with the player icons, used to indicate it is the player’s turn.

PlayerTurnTextSprite

Declaration

```
public Sprite playerTurnTextSprite;
```

Field Value

Type	Description
Sprite	The sprite image with the player turn text.

RoundCounter

Declaration

```
private int _roundCounter = 1;
```

Field Value

Type	Description
Int	Numerical value indicating the number of rounds the game has gone through.

RoundCounterText

Declaration

```
public Text roundCounterText;
```

Field Value

Type	Description
Text	UI text component used to indicate what round we are on.

State

Declaration

```
public BattleState state;
```

Field Value

Type	Description
BattleState	Field indicating what is currently happening in the combat system.

UnitsAlive

Declaration

```
private List<Humanoid> unitsAlive = new List<Humanoid>( );
```

Field Value

Type	Description
List<Humanoid>	Private list that is populated with all players and enemies that are enabled in the scene.

Remarks

This list is populated with all units that are alive.

Even enemies that are enabled in the scene, but are currently hidden by fog, are also added into the list.

Enums

ActiveUnits

Detail

Determines which side is currently active in the combat system.

Syntax

```
public enum ActiveUnits
```

Fields

Name	Description
Players	Indicates it is the player’s turn.
Enemies	Indicates it is the enemy’s turn.

Attack

Detail

Determines what type of attack is being executed.

Syntax

```
private enum Attack
```

Fields

Name	Description
NormalAttack	The selected player unit’s normal attack was executed.
AbilityOne	The selected player unit’s first ability was executed.
AbilityTwo	The selected player unit’s second ability was executed.

BattleState

Detail

Determines what kind of state the combat system is in.

Syntax

```
public enum BattleState
```

Fields

Name	Description
Start	Indicates it is the start up process of the scene.
Idle	Indicates that the system is in an idle state, no actions are being performed.
PerformingAction	Indicates something is happening in the system (i.e. animation being played).
Targeting	Indicates that a player unit is targeting an ability or attack.
Won	Indicates that the player has won the level.
Lost	Indicates that the player has lost the level.

Functions

AbilityOne

Description

Begins the targeting process for the selected player unit’s first ability.

Declaration

```
public void AbilityOne( )
```

Remarks

Code is executed based on OnClick function of a Unity button.

AbilityTwo

Description

Begins the targeting process for the selected player unit’s second ability.

Declaration

```
public void AbilityTwo( )
```

Remarks

Code is executed based on OnClick function of a Unity button.

ActivateCombatButtons

Description

Activates the combat buttons on the overlay.

Declaration

```
public void ActivateCombatButtons( )
```

AllDefend

Description

Forces all player units that can still perform an action to defend this round.

Declaration

```
public void AllDefend( )
```

Remarks

Code is executed based on OnClick function of a Unity button.

AttackComplete

Description

Indicates that the selected player unit has finished performing their attack.

Declaration

```
public void AttackComplete( )
```

Remarks

The code is executed through a System.Action delegate invoke.

Cancel

Description

Cancels the action we are currently targeting.

Declaration

```
public void Cancel(bool deselectPlayer = true)
```

Parameters

Type	Name	Description
bool	deselectPlayer	Indicates if we also want the player to be deselected when hitting cancel.

Remarks

The Boolean is set in inspector window in the OnClick function as true.

The player unit will only be deselected if they are not currently targeting and the combat system is in an idle state.

CheckAreaCondition

Description

Checks to see the player units are within the target victory zone.

Declaration

```
public bool CheckAreaCondition(ObjectiveZone zone)
```

Parameters

Type	Name	Description
ObjectiveZone	zone	TODO

Returns

Type	Description
bool	If player is in the target zone a true value will be returned, otherwise it will return false.

CheckKillCondition

Description

Checks to see if we have killed all of the specified enemy units to win the level.

Declaration

```
public bool CheckKillCondition(EnemyType typeToKill)
```

Parameters

Type	Name	Description
EnemyType	typeToKill	Indicates the enemy type to search for in the list.

Returns

Type	Description
bool	If there are no more enemy units of the specified type alive a value of true is returned, otherwise false.

CheckLoseCondition

Description

Checks to see if there are any remaining player units that are alive.

Declaration

```
private bool CheckLoseCondition( )
```

Returns

Type	Description
bool	Returns false if there is still a player unit that's alive in the scene, otherwise true.

DeactivateCombatButtons

Description

Deactivates the combat buttons.

Declaration

```
public void DeactivateCombatButtons( )
```

Defend

Description

Causes the selected player unit to defend this round.

Declaration

```
public void Defend( )
```

EndUnitTurn

Description

Ends the turn for the current unit, removing them from their respective list.

Declaration

```
private void EndUnitTurn(Humanoid unit)
```

Parameters

Type	Name	Description
Humanoid	unit	The unit whose turn is over.

EnemyTurn

Description

Performs the logic for the enemy units on their turn.

Declaration

```
IEnumerator EnemyTurn( )
```

Remarks

Goes through the entire EnemiesToGo list until all of the currently revealed enemies have performed their actions this round.

If an enemy is within the list, but is not revealed, that enemy will automatically be removed, and the code will continue.

GameLost

Description

Activates the end game canvas and changes the text to display “You Lose!”

Declaration

```
private void GameLost( )
```

GameWon

Description

Activates the end game canvas and changes the text to display “You Win!”

Declaration

```
private void GameWon( )
```

HideAbilityInfo

Description

Hides the popup explaining the ability of the selected player unit.

Declaration

```
public void HideAbilityInfo( )
```

KillUnit

Description

Destroys the killed unit and updates the system according to match the current state.

Declaration

```
public void KillUnit(Humanoid unit)
```

Parameters

Type	Name	Description
Humanoid	unit	The unit whose health is at or below 0.

KillUnitCR

Description

Future coroutine that will affect how units are killed, allowing their death audio to play out until the end before they are destroyed.

Declaration

```
IEnumerator KillUnitCR(Humanoid unit)
```

Parameters

Type	Name	Description
Humanoid	unit	The unit whose health is at or below 0.

Remarks

This coroutine in its current state is dependent on the killed unit having audio files to play. For the time being it will remain unused until more audio has been accumulated.

NewRound

Description

Starts a new round in the combat system.

Declaration

```
private void NewRound( )
```

NewSpawn

Description

Spawns a new enemy and adds them to the UnitsAlive list.

Declaration

```
public void NewSpawn(Humanoid spawn)
```

Parameters

Type	Name	Description
Humanoid	spawn	TODO

NormalAttack

Description

Begins the targeting process for the selected player unit’s normal attack.

Declaration

```
public void NormalAttack( )
```

Remarks

Code is executed based on the OnClick function of a Unity button.

ProcessAttack

Description

Executes the requested attack/ability on the currently selected player unit.

Declaration

```
private void ProcessAttack(Attack type)
```

Parameters

Type	Name	Description
Attack	type	The attack/ability of the selected player to execute.

SetAbilityInfo

Description

Sets the sprite for the ability info popup window based on the name of the button the cursor is over.

Declaration

```
public void SetAbilityInfo(Button button)
```

Parameters

Type	Name	Description
Button	button	The button whose name needs to be checked.

SetAbilityOneButtonState

Description

Sets the ability one button’s interactable state.

Declaration

```
public void SetAbilityOneButtonState(bool activeState)
```

Parameters

Type	Name	Description

bool	activeState	Indicates if we want the button on or off.

SetAbilityTwoButtonState

Description

Sets the ability two button's interactable state.

Declaration

```
public void SetAbilityTwoButtonState(bool activeState)
```

Parameters

Type	Name	Description
bool	activeState	Indicates if we want the button on or off.

Set Active Units

SetActiveUnits

Description

Sets the value of the ActiveUnits field.

Declaration

```
public void SetActiveUnits(ActiveUnits activeUnits)
```

Parameters

Type	Name	Description
ActiveUnits	activeUnits	The units that are going to be active.

SetBattleState

Description

Sets the value of the State field.

Declaration

```
public void SetBattleState(BattleState state)
```

Parameters

Type	Name	Description
BattleState	state	The new state of the combat system.

SetCoolDownText

Description

Sets the cooldown text for each ability of the selected player unit.

Declaration

```
public void SetCoolDownText(Player player)
```

Parameters

Type	Name	Description
Player	player	The player to compare to selected and last selected.

SetEnemyCountText

Description

Sets the text representing the number of enemies alive.

Declaration

```
public void SetEnemyCountText( )
```

SetTurnUI

Description

Updates the UI to represent the currently active side.

Declaration

```
private void SetTurnUI(ActiveUnits activeSide)
```

Parameters

Type	Name	Description
ActiveUnits	activeSide	The side that is going to be active.

SetUpBattle

Description

Sets up the map and combat system with the necessary info.

Declaration

```
private void SetupBattle( )
```

SubscribeEnemy

Description

Adds a revealed enemy to the turn system.

Declaration

```
public void SubscribeEnemy(Enemy enemy)
```

Parameters

Type	Name	Description
Enemy	enemy	The enemy to add to the list.

SubscribeTimerUnit

Description

Subscribes a unit that has been buffed or debuffed to the system. At the end of each round these units will have their timers updated.

Declaration

```
public void SubscribeTimerUnit(Humanoid subject)
```

Parameters

Type	Name	Description
Humanoid	subject	The unit that is to be added.

UnsubscribeTimerUnit

Description

Removes a unit from the timer list when they are killed.

Declaration

```
public void UnsubscribeTimerUnit(Humanoid subject)
```

Parameters

Type	Name	Description
Humanoid	subject	The unit that is to be removed.

Remarks

Yes, I know this is weird and that it makes no sense how I structured it. Sue me.

UpdateTimers

Description

Advances the timer of all of the units in the TimerUnits list.

Declaration

```
private void UpdateTimers( )
```

Class Humanoid

Fields

ActiveParticle

Declaration

```
protected ParticleSystem activeParticle;
```

Field Value

Type	Description
Particle System	The currently active particle system on the unit.

AnimatorController

Declaration

```
public Animator animatorController;
```

Field Value

Type	Description
Animator	The animator controller for this unit’s rig.

AttackParticle

Declaration

```
public ParticleSystem attackParticle;
```

Field Value

Type	Description
Particle System	The particle system for this unit’s normal attack.

Remarks

Typically, this is only for the player units, but it is possible we will have enemy attack particles in the future.

AttackRange

Declaration

```
private int _attackRange;
```

Field Value

Type	Description
int	The private variable for the normal attack range of this unit.

AttackShape

Declaration

```
public ActionShape AttackShape = ActionShape.Diamond;
```

Field Value

Type	Description
ActionShape	The attack range shape for this unit.

BaseAttack

Declaration

```
protected int _baseAttack;
```

Field Value

Type	Description
int	The default base attack stat of the unit. Helpful for resetting stats.

BaseDefense

Declaration

```
protected int _baseDefense;
```

Field Value

Type	Description
int	The default base defense stat of the unit. Helpful for resetting stats.

BaseMovement

Declaration

```
protected int _baseMovement;
```

Property Value

Type	Description
int	The default base movement stat of the unit. Helpful for resetting stats.

BaseRange

Declaration

```
protected int _baseRange;
```

Field Value

Type	Description
int	The default attack range of the unit. Helpful for resetting stats.

BaseStats

Declaration

```
[SerializeField] private CharacterStats _baseStats;
```

Field Value

Type	Description
CharacterStats	The container of the base stats for the unit.

CurrentTile

Declaration

```
public Tile currentTile;
```

Field Value

Type	Description
Tile	The tile that the unit is currently occupying.

DamagedThisTurn

Declaration

```
public bool damagedThisTurn = false;
```

Field Value

Type	Description
bool	Indicates if the unit was damaged this round.

DamageText

Declaration

```
public text damageText;
```

Field Value

Type	Description
Text	The text component representing how much damage was dealt to this unit.

DefendParticle

Declaration

```
public ParticleSystem defendParticle;
```

Field Value

Type	Description
Particle System	The particle system that is played when this unit defends this round.

Health

Declaration

```
private int _health;
```

Field Value

Type	Description
int	The health of the unit. For accessing the health of the unit refer to the Health property instead.

HealthBar

Declaration

```
public Slider healthBar;
```

Field Value

Type	Description
Slider	The graphical slider representing our health bar.

HealthText

Declaration

```
public Text healthText;
```

Field Value

Type	Description
Text	The text component representing this unit’s health in numerical form.

MaxHealth

Declaration

```
protected int _maxHealth;
```

Field Value

Type	Description

int	The maximum health of the unit. Prevents the player's health from going over the maximum when they are healed by an outside source.

MoveSpeedModifier

Declaration

```
protected int moveSpeedModifier;
```

Field Value

Type	Description
int	Modifies the player unit's speed when there are no enemies visible in the level.

Remarks

An enemy being visible means that it is not hidden by fog and it's revealed field is set to true.

Moving

Declaration

```
private bool moving = false;
```

Field Value

Type	Description
bool	Indicates if a unit is currently moving.

ParentTransform

Declaration

```
public Transform parentTransform;
```

Field Value

Type	Description
Transform	Refers to the parent game object of the script.

Remarks

The parent transform will be assigned automatically in the Start function for each unit.

The parent game object will need to be marked with the “unit holder” tag within the inspector.

If there is no parent game object, meaning the prefab hasn’t been updated with any animations yet to warrant a change, then the parent transform will be set with a reference to the object containing the script instead.

RemainingActions

Declaration

```
public int RemainingActions = 2;
```

Field Value

Type	Description
int	Refers to how many actions the unit has left this round. Useful for overriding the system if necessary.

StatusEffects

Declaration

```
protected List<StatusEffect> statusEffects = new Lit<StatusEffect>( );
```

Field Value

Type	Description
List<StatusEffect>	Contains references to all of the status effects currently active on this unit.

TileCrossTime

Declaration

```
public float tileCrossTime = 0.3f;
```

Field Value

Type	Description
float	The time it takes for a unit to move from one tile to another.

TurnSmoothTime

Declaration

```
private float turnSmoothTime = 0.1f;
```

Field Value

Type	Description
float	The it takes to switch directions.

TurnSmoothVelocity

Declaration

```
private float turnSmoothVelocity;
```

Field Value

Type	Description
float	TODO

UnitAudio

Declaration

```
public UnitAudioPlayer unitAudio;
```

Field Value

Type	Description
UnitAudioPlayer	A reference to the container which holds and plays all audio files that are used for combat and are unique to each unit.

Properties

AnimationComplete

Declaration

```
public bool AnimationComplete { get; set; } = false;
```

Property Value

Type	Description
bool	Indicates that a certain frame of the animation has been reached, allowing code to continue.

Remarks

This allows for coroutines to continue executing their code, specifically for damage output and ability behavior, once certain frames of the animation have been reached, allowing for cleaner visuals that match up to what we expect to see in game.

AttackRange

Declaration

```
public int AttackRange { get; set; }
```

Property Value

Type	Description
int	The range of the normal attack for this unit.

Remarks

Returns the value of the _attackRange field.

Automatically clamps the value of the attack range to a maximum value of ten.

AttackStat

Declaration

```
public int AttackStat { get; set; }
```

Property Value

Type	Description
int	The attack stat of the unit.

AttackTileRange

Declaration

```
public bool [ , ] AttackTileRange { get; set; }
```

Property Value

Type	Description
bool[,]	The tiles that the unit can currently reach with its attack.

DefenseStat

Declaration

```
public int DefenseStat { get; set; }
```

Property Value

Type	Description
int	The defense stat of the unit.

DefendState

Declaration

```
public DefendingState DefendState { get; set; }
```

Property Value

Type	Description
DefendingState	States whether or not the unit is defending this round.

DexterityStat

Declaration

```
public float DexterityStat { get; set; }
```

Property Value

Type	Description
float	The dodge chance of the unit.

HasAttacked

Declaration

```
public bool HasAttacked { get; set; }
```

Property Value

Type	Description
bool	Indicates that this unit has executed an attack this round.

HasMoved

Declaration

```
public bool HasMoved { get; set; }
```

Property Value

Type	Description
bool	Indicates that this unit has executed a move this round.

Health

Declaration

```
public int Health { get; set; }
```

Property Value

Type	Description
int	The current health of the unit.

Remarks

Automatically updates the UI elements that represent the unit's health.

Value of health is clamped between 0 and the value assigned to max health.

IsTurning

Declaration

```
protected bool IsTurning { get; set; } = false;
```

Property Value

Type	Description
bool	Indicates that the unit is turning.

MaxHealth

Declaration

```
public int MaxHealth { get; }
```

Property Value

Type	Description
int	The max health of this unit.

MovementStat

Declaration

```
public int MovementStat { get; set; }
```

Property Value

Type	Description
int	The movement range of the unit.

State

Declaration

```
public HumanoidState State { get; set; }
```

Property Value

Type	Description
HumanoidState	The state of the unit at that current moment.

Remarks

As of this current moment the only state that has any weight in the system is HumanoidState.Moving. It is possible to make the other states actually prevent code from executing but as of this moment it’s just the moving state.

TileRange

Declaration

```
public bool[ , ] TileRange { get; set; }
```

Property Value

Type	Description
bool [,]	Tiles the unit can move to.

XpDrop

Declaration

```
public int XpDrop { get; set; }
```

Property Value

Type	Description
int	The amount of xp that is dropped when this unit dies.

Enums

ActionShape

Detail

Indicates the shape of the action that will be performed. The shape of the range.

Syntax

```
public enum ActionShape
```

Fields

Name	Description
Cross	The action will be displayed in the shape of a cross.
Diamond	The action will be displayed in the shape of a diamond. Does not move around corners.
Flood	The action will be displayed in the shape of a diamond. Can move around corners. Ideal for movement.
Square	The action will be displayed in the shape of a square.

DefendingState

Detail

Indicates if the unit is within a defending state or not.

Syntax

```
public enum DefendingState
```

Fields

Name	Description
NotDefending	The unit is not defending this round.
Defending	The unit is defending this round.

HumanoidState

Detail

Represents the current state in the system.

Syntax

```
public enum HumanoidState
```

Fields

Name	Description
Idle	The unit is currently not doing anything.
Selected	The unit is currently selected in the character selector.
Moving	The unit is currently moving around on the map
Targeting	The unit is currently targeting an attack.
Attacking	The unit is currently attacking.
Done	The unit is done for this round.

Functions

ActivateAttackParticle

Description

Activates the attack particle system if it exists.

Declaration

```
protected void ActivateAttackParticle( )
```

AddStatusEffect

Description

Adds a new status effect to the list of status effects active on this unit.

Declaration

```
public virtual void AddStatusEffect(StatusEffect effect)
```

Parameters

Type	Name	Description
StatusEffect	effect	The status effect to be added.

AdvanceTimer

Description

Advances the timer on the unit’s status effects.

Declaration

```
public virtual void AdvanceTimer( )
```

CheckForEffectOfType

Description

Searches through the list of status effects to see if this unit has a status effect of a certain type on them currently.

Declaration

```
protected bool CheckForEffectOfType(StatusEffect.StatusEffectType type)
```

Parameters

Type	Name	Description
StatusEffect.StatusEffectType	type	The type of status effect to search for.

Returns

Type	Description
bool	True if the type of status effect was found, false otherwise

Defend

Description

Causes the unit to defend, temporarily raising their defense stat.

Declaration

```
public virtual void Defend( )
```

FindMovementRange

Description

Finds the movement range of the unit at their current position.

Declaration

```
public void FindMovementRange( )
```

Remarks

Changes the value assigned to the tile range array.

GetNumOfStatusEffects

Description

Returns the number of status effects active on this unit.

Declaration

```
public int GetNumOfStatusEffects( )
```

Returns

Type	Description
int	The number of status effects active on this unit.

GetSourceOfStatusEffect

Description

Returns the source of the status effect.

Declaration

```
public Humanoid GetSourceOfStatusEffect(StatusEffect.StatusEffectType type)
```

Parameters

Type	Name	Description
StatusEffect.StatusEfectType	type	The type of status effect to search for.

Returns

Type	Description
Humanoid	A reference to the source of the status effect.

HealingTileCheck

Description

Checks to see if the player is on a healing tile.

Declaration

```
private void HealingTileCheck( )
```

LookInDirection

Description

Smoothly turns the unit towards the direction.

Declaration

```
protected bool LookInDirection(Vector3 direction)
```

Parameters

Type	Name	Description
Vector3	direction	The direction we want to look in.

Returns

Type	Description
bool	True if we are still turning, false otherwise.

LookToTarget

Description

Coroutine that turns the unit in the direction of their target.

Declaration

```
protected virtual IEnumerator LookToTarget( )
```

Move

Description

Begins the movement coroutine for moving on the map.

Declaration

```
public virtual void Move(List<Tile> path, bool bypassRangeCheck = false)
```

Parameters

Type	Name	Description
List<Tile>	path	The path the unit will take.
bool	bypassRangeCheck	Uhhhhhh

MoveCR

Description

Movement coroutine that moves the unit along the grid.

Declaration

```
IEnumerator MoveCR(List<Tile> path)
```

Parameters

Type	Name	Description
List<Tile>	path	The path we wish to follow.

PlayAudio

Description

Makes a reference to this unit’s audio player and plays the necessary clip.

Declaration

```
public void PlayAudio(UnitAudioPlayer.AudioToPlay toPlay)
```

Parameters

Type	Name	Description
AudioToPlay	toPlay	The audio clip to play.

ResetSpecificStat

Description

Resets a specific stat on the unit.

Declaration

```
public void ResetSpecificStat(StatusEffect.StatusEffectType stat)
```

Parameters

Type	Name	Description
StatusEffectType	stat	The stat to reset.

SetActiveParticle

Description

Sets the active particle that we wish to execute.

Declaration

```
public void SetActiveParticle(ParticleSystem particle)
```

Parameters

Type	Name	Description
ParticleSystem	particle	The particle system we want to play.

SetAnimationComplete

Description

Sets the animation complete property.

Declaration

```
public void SetAnimationComplete(bool value)
```

Parameters

Type	Name	Description
bool	value	Animation is complete or not.

ShowDamage

Description

Displays the damage text for a short time.

Declaration

```
IEnumerator ShowDamage(int damage, bool blocked = false)
```

Parameters

Type	Name	Description
int	damage	The amount of damage dealt.
bool	blocked	Indicates if the attack was blocked

TakeDamage

Description

Deals damage to the unit.

Declaration

```
public bool TakeDamage(int damage, bool trueDamage = false)
```

Parameters

Type	**Name**	**Description**
int	damage	The amount of damage to deal.
bool	trueDamage	Indiactes if the attacking unit has true damage. (i.e. The Archer)

Class Player

Fields

Ability1Sprites

Declaration

```
public Sprite[ ] Ability1Sprites = new Sprite[5];
```

Field Value

Type	Description
Sprite[ ]	The sprites of the player's first ability.

Ability2Sprites

Declaration

```
public Sprite[ ] Ability2Sprites = new Sprite[5];
```

Field Value

Type	Description
Sprite[ ]	The sprites of the player's second ability.

AbilityOneCooldown

Declaration

```
public int AbilityOneCooldown;
```

Field Value

Type	Description
int	The cooldown of the player's first ability.

AbilityOneParticle

Declaration

```
public ParticleSystem AbilityOneParticle
```

Field Value

Type	Description
ParticleSystem	The particle system for the player’s first ability.

AbilityOneRange

Declaration

```
public int AbilityOneRange
```

Field Value

Type	Description
int	Range of player’s first ability.

AbilityTwoCooldown

Declaration

```
public int AbilityTwoCooldown
```

Field Value

Type	Description
int	The cooldown of the player’s second ability.

AbilityTwoParticle

Declaration

```
public ParticleSystem AbilityTwoParticle
```

Field Value

Type	Description
ParticleSystem	The particle system for the player’s second ability.

AbilityTwoRange

Declaration

```
public int AbilityTwoRange
```

Field Value

Type	Description
int	The range of the player’s second ability.

CurrentObjectiveZone

Declaration

```
ObjectiveZone currentObjectiveZone = null
```

Field Value

Type	Description
ObjectiveZone	Objective zone the player currently occupies.

ExpParticle

Declaration

```
public ParticleSystem ExpParticle
```

Field Value

Type	Description
ParticleSystem	Exp particle system that is a child of the player unit.

NormalAttackSprites

Declaration

```
public Sprite[ ] NormalAttackSprites = new Sprite[5]
```

Field Value

Type	Description
Sprite[ ]	The sprites of the player’s normal attack.

RemainingAbilityOneCooldown

Declaration

```
int _remainingAbilityOneCD
```

Field Value

Type	Description
int	The remaining cooldown on ability one.

RemainingAbilityTwoCooldown

Declaration

```
int _remainingAbilityTwoCD
```

Field Value

Type	Description
int	The remaining cooldown on ability two.

Selected

Declaration

```
bool selected = false;
```

Field Value

Type	Description
bool	Indicates if the player is selected.

SelectedParticle

Declaration

```
public ParticleSystem SelectedParticle
```

Field Value

Type	Description
ParticleSystem	The particle system that is played when the player is selected.

UpgradeToggleSprites

Declaration

```
public Sprite[ ] UpgradeToggleSprites = new Sprite[3]
```

Field Value

Type	Description
Sprite[ ]	TODO

Properties

AbilityOneTileRange

Declaration

```
public bool[ , ] AbilityOneTileRange { get; set; }
```

Property Value

Type	Description
bool[,]	Tile range of the player’s first ability.

AbilityTwoTileRange

Declaration

```
public bool[ , ] AbilityTwoTileRange { get; set; }
```

Property Value

Type	Description
bool[,]	Tile range of the player’s second ability.

RemainingAbilityOneCD

Declaration

```
public int RemainingAbilityOneCD { get; }
```

Property Value

Type	Description

int	The remaining cooldown on ability one.

RemainingAbilityTwoCD

Declaration

```
public int RemainingAbilityTwoCD { get; ]
```

Property Value

Type	Description
int	The remaining cooldown on ability two.

Functions

AbilityOne

Description

Abstract function for player ability one.

Declaration

```
public abstract void AbilityOne(Action callback)
```

Parameters

Type	Name	Description
System.Action	callback	The function to call when logic is completed.

AbilityOneAnim

Description

Triggers the ability one animation for this player.

Declaration

```
protected void AbilityOneAnim( )
```

AbilityOneCR

Description

Ability one coroutine.

Declaration

```
protected abstract IEnumerator AbilityOneCR(Action callback);
```

Parameters

Type	Name	Description
System.Action	callback	The function to call when logic is completed.

AbilityOneUpgradeOne

Description

Abstract function for upgrading ability one.

Declaration

```
protected abstract void AbilityOneUpgradeOne( );
```

AbilityOneUpgradeTwo

Description

Abstract function for upgrading ability one.

Declaration

```
protected abstract void AbilityOneUpgradeTwo( );
```

AbilityTwo

Description

Abstract method for activating player’s second ability.

Declaration

```
public abstract void AbilityTwo(Action callback);
```

Parameters

Type	Name	Description
System.Action	callback	The function to be called when coroutine is finished.

AbilityTwoAnim

Description

Triggers the ability two animation for this player.

Declaration

```
protected void AbilityTwoAnim( )
```

AbilityTwoCR

Description

Ability two coroutine

Declaration

```
public abstract void AbilityTwo(Action callback)
```

Parameters

Type	Name	Description
System.Action	callback	The function to call when logic is completed.

AbilityTwoUpgradeOne

Description

Abstract function for upgrading ability two.

Declaration

```
protected abstract void AbilityTwoUpgradeOne( );
```

AbilityTwoUpgradeTwo

Description

Abstract function for upgrading ability two.

Declaration

```
protected abstract void AbilityTwoUpgradeTwo( );
```

ActivateAbilityOneParticle

Description

Activates the particle effect for ability one.

Declaration

```
public void ActivateAbilityOneParticle( )
```

ActivateAbilityTwoParticle

Description

Activates the particle effect for ability two.

Declaration

```
public void ActivateAbilityTwoParticle( )
```

AdvanceTimer

Description

Override of advance timer in humanoid that also reduces the cooldown on abilities.

Declaration

```
public override void AdvanceTimer( )
```

AttackAnim

Description

Executes the normal attack animation of this player.

Declaration

```
protected void AttackAnim( )
```

AttackUpgradeOne

Description

Abstract function for upgrading the normal attack.

Declaration

```
protected abstract void AttackUpgradeOne( );
```

AttackUpgradeTwo

Description

Abstract function for upgrading the normal attack.

Declaration

```
protected abstract void AttackUpgradeTwo( );
```

AtTargetObjectives

Description

Checks to see if the player is at the target zone.

Declaration

```
public bool AtTargetObjective(ObjectiveZone target)
```

Parameters

Type	Name	Description
ObjectiveZone	target	TODO

Returns

Type	Description
bool	True if we are at objective zone, false otherwise.

DeactivateAbilityOneParticle

Description

Deactivates the particle system for the player’s first ability.

Declaration

```
protected void DeactivateAbilityOneParticle( )
```

DeactivateAbilityTwoParticle

Description

Deactivates the particle system for the player’s second ability.

Declaration

```
protected void DeactivateAbilityTwoParticle( )
```

DoubleMoveSpeed

Description

Doubles the move speed of the player.

Declaration

```
public void DoubleMoveSpeed( )
```

FindActionRanges

Description

Find the tile range of the player’s actions.

Declaration

```
public void FindActionRanges( )
```

GetTargetPos

Description

Returns a vector3 representation of the target’s position.

Declaration

```
protected Vector3 GetTargetPos( )
```

Returns

Type	Description
Vector3	The target’s position.

Heal

Description

Heals the player.

Declaration

```
public void Heal( )
```

InObjectiveZone

Description

Sets currentObjectiveZone to zone entered or null if zone exited.

Declaration

```
public void InObjectiveZone(ObjectiveZone zone, bool entered)
```

Parameters

Type	Name	Description
ObjectiveZone	zone	TODO
bool	entered	

Move

Description

Override of the Move function in Humanoid. Triggers the walking animation of the player.

Declaration

```
public override void Move(List<Tile> path, bool bypassRangeCheck = false)
```

Parameters

Type	Name	Description
List<Tile>	path	The path to traverse.
bool	bypassRangeCheck	UHHHHHHH

NormalAttack

Description

Abstract function for activating the normal attack.

Declaration

```
public abstract void NormalAttack( );
```

NormalAttackCR

Description

Abstract function for the normal attack coroutine. Used for allowing players to target.

Declaration

```
protected abstract IEnumerator NormalAttackCR(Action callback);
```

Parameters

Type	Name	Description
System.Action	callback	The function to call when logic is completed.

ProcessUpgrade

Description

Abstract function that reads the upgrade that was purchased and activates the code for it.

Declaration

```
public abstract void ProcessUpgrade(Abilities abilityToUpgrade);
```

Parameters

Type	Name	Description
Abilities	abilityToUpgrade	The ability that’s being upgraded.

SetMoveSpeedNormal

Description

Resets the speed modifier back to zero.

Declaration

```
public void SetMoveSpeedNormal( );
```

ShowHealText

Description

Shows the amount of health healed above the player.

Declaration

```
public IEnumerator ShowHealText(int amount)
```

Parameters

Type	Name	Description
int	amount	The amount of health healed.

StartAbilityOneCD

Description

Starts the cooldown of the player’s first ability.

Declaration

```
protected void StartAbilityOneCD( )
```

StartAbilityTwoCD

Description

Starts the cooldown of the player’s second ability.

Declaration

```
protected void StartAbilityTwoCD( )
```

UnitDeselected

Description

Deactivates all coroutines and hides information specific to that unit when they are deselected.

Declaration

```
public void UnitDeselected( )
```

UnitSelected

Description

Sets up the game system with data specific to the player unit.

Declaration

```
public void UnitSelected( )
```

Class Send Message On Event

Enums

ParticleToTrigger

Detail

Enum representing the particle that is to be triggered and played.

Syntax

```
public enum ParticleToTrigger
```

Fields

Name	Description
Fireball	The mage’s fireball projectile.

Remarks

Currently this enum only represents the mage's fireball. The purpose of this enum is meant as a reminder for future Chase if the project is approved for 495 Gold, at which point there is the possibility of more player character's some of which could have their own projectiles as well that are primarily particle systems.

ProjectileToActivate

Detail

Enum representing the projectile that needs to be activated.

Syntax

```
public enum ProjectileToActivate
```

Fields

Name	Description
Archer_Arrow	The archer’s arrow projectile.
Archer_Potion	The archer’s potion projectile.

Remarks

This function is typically used for the projectiles that are only made up of particle systems and need to executed at certain frames of the animation, such as the fireball.

Fields

Name

Declaration

Field Value

Type	Description

Properties

Name

Declaration

Property Value

Type	Description

Functions

Name

Description

Declaration

Parameters

Type	Name	Description

Returns

Type	Description

Enums

Name

Detail

Syntax

```
```

Fields

Name	Description