InfoEducație 2014

Dark Energy

Software Architect
Andrei Muntean

Game Artist
Diana Ghinea

GAME GUIDE

INTRODUCTION

Dark Energy is a turn-based role-playing game (RPG) designed for mobile devices. It allows the player to assume the role of a heroic character and explore a virtual world based in a high fantasy setting.

In a universe imbued with mystery and magic, **Dark Energy** is the enigmatic force that gives rise to all supernatural events. At the heart of every spirit, arcane entity, spell and incantation lies Dark Energy.

CHARACTER PROGRESSION

Dark Energy is centered on character development. While traveling, questing or fighting monsters, the hero acquires **Experience Points**. When sufficient experience has been gathered, the character **levels up**, gaining certain points that can be used to increase its power.

Attribute Points
Used to increase the attributes of the hero.

Mastery Points
Used to upgrade the abilities of the hero.

Attributes

Attributes are the basic building blocks of every character in the game.

Strength
Increases defense and physical damage.

Intuition
Increases magical power.

Reflexes
Increases the chance to evade an attack. The most reflexive units attack first during combat.

Vitality
Increases health.

Vigor
Increases the amount of **Dark Energy** generated.

Weapon Damage and **Armor** represent an additional layer of attributes that depends entirely on the items equipped by the character.

Elements

There are eight elements in Dark Energy and they are intertwined with one another.

Characters have an offensive element with which they attack and a defensive one that acts as a shield.

Light
Highly effective against darkness. Effective against earth and water. Weak against air and shock.

Air
Highly effective against earth. Effective against fire and darkness. Weak against ice and light.

Ice
Highly effective against fire. Effective against shock and earth. Weak against water and air.

Water
Highly effective against shock. Effective against light and fire. Weak against darkness and ice.

Darkness
Highly effective against light. Effective against air and shock. Weak against earth and water

Earth
Highly effective against air. Effective against ice and light. Weak against fire and darkness.

Fire
Highly effective against ice. Effective against water and air. Weak against shock and earth.

Shock
Highly effective against water. Effective against darkness and ice. Weak against fire and light.

Abilities

Characters use abilities to interact with the world. Abilities can be beneficial, such as a healing spell or aura of protection; or harmful, e.g. a physical strike or fire bolt. Most abilities consume Dark Energy when used.

Inventory

Equipment

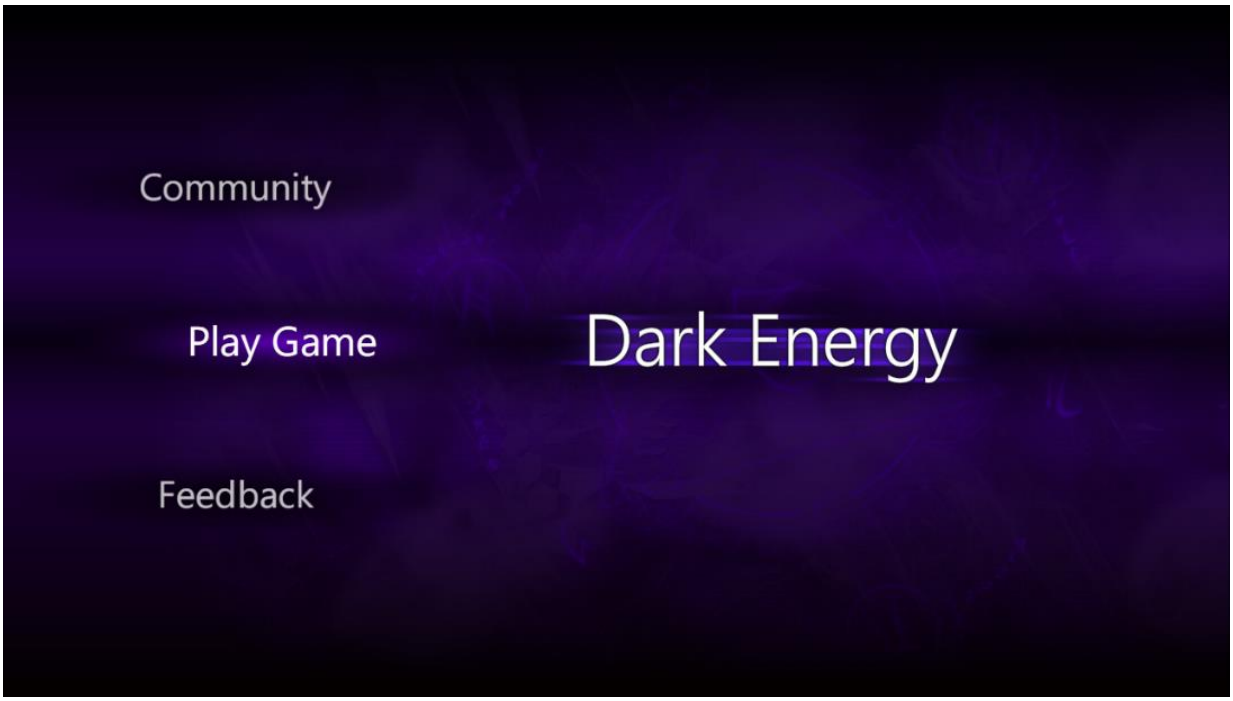
Heroes can equip weapons and armor to become more powerful. An item can be equipped for each of the following slots: **Weapon** (defines the **Offensive Element**), **Relic** (defines the **Defensive Element**), **Head**, **Neck**, **Chest**, **Back**, **Hands**, **Finger**, **Legs** and **Feet.**

Miscellaneous

Characters gain **Gold** by winning battles and trading. Gold can be used to purchase most items in the game. **Dark Crystals** represent a rarer form of currency that can be traded for special items.

STARTING THE GAME

Upon entering the game, the user is greeted by the main menu. Pressing the upper half of the screen allows the user to cycle to the previous option, while pressing the lower half of the screen selects the next option.



Tapping **Community** directs the user to the official Facebook page of Dark Energy. The page will become available shortly before the game is released.

Tapping **Play Game** sends the user to the **Character Selection** screen.

Tapping **Feedback** allows the user to review the application in the Windows Phone Store. This option will be functional once the game is released.

CREATING A CHARACTER

At the **Character Selection** menu, the user can create and delete characters. Selecting an empty slot and tapping **CREATE** sends the player to the **Character Creation** screen.

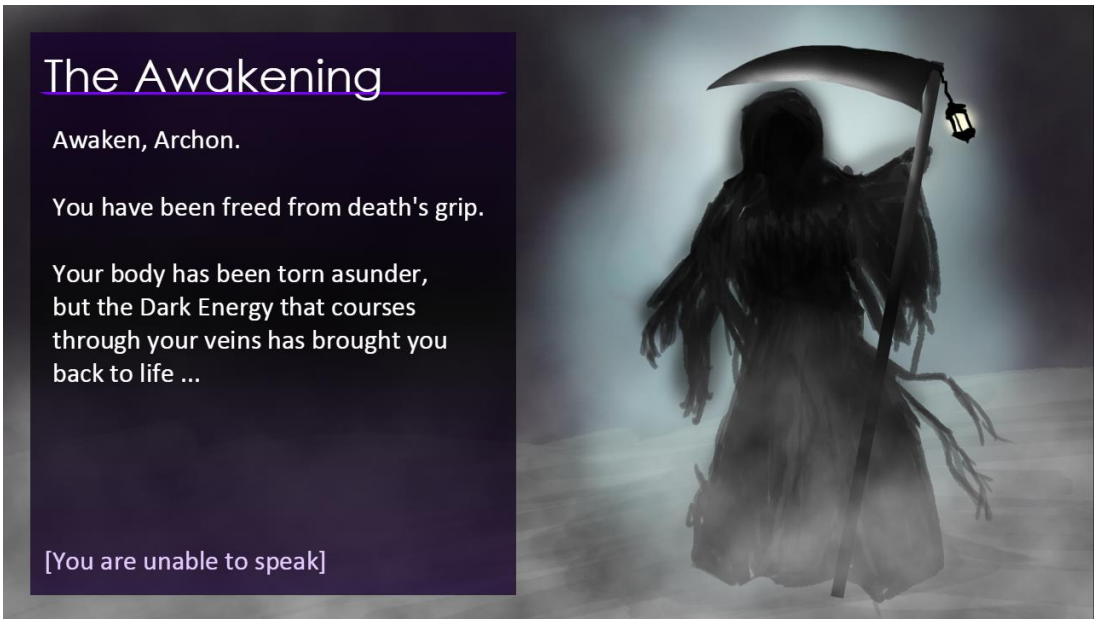
The **Character Creation** menu allows the user to create its own hero, as shown below.

A player can change the **gender**, **skin color**, **facial features** and **hairstyle** of the character. By tapping the box above the character, the user can assign a **name** to the hero.

Tapping **CREATE** finalizes the character creation process.

EMBARKING ON AN EPIC JOURNEY

Once a character has been created, the player can tap **PLAY** to enter the world. This prompts the start of a cinematic scene, which the user navigates through by pressing anywhere on the screen. An enigmatic figure awakens you, as seen below.



Once the cinematic ends, the player finds itself in a town known as **Westhill**.

The world can be explored by tapping the regions that the map markers point at. The buttons located at the bottom of the screen represent the user interface.

CHARACTER MENU

The character menu allows the player to view hero data, distribute attribute points and upgrade abilities. It is divided into three categories.

General

Displays the name, level, attributes, elements and experience of the hero.

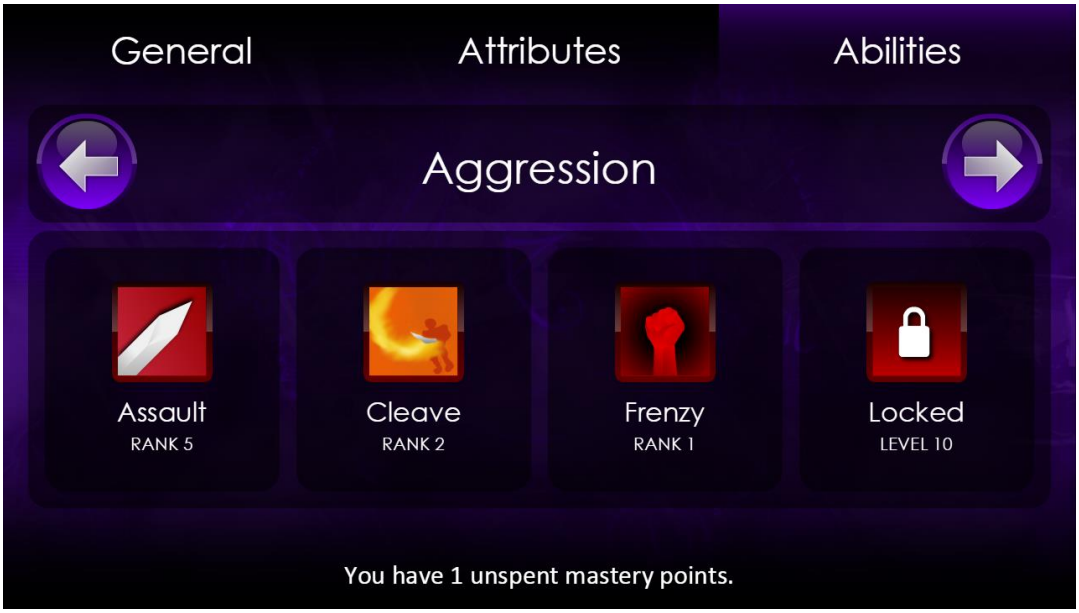
Attributes

Shows detailed information about character attributes and allows the player to improve them.

Attributes can be improved by tapping the buttons on the left panel. Changes are displayed in real time on the right panel. Once the user has decided on a configuration, the modifications can be saved by tapping **Confirm Changes**.

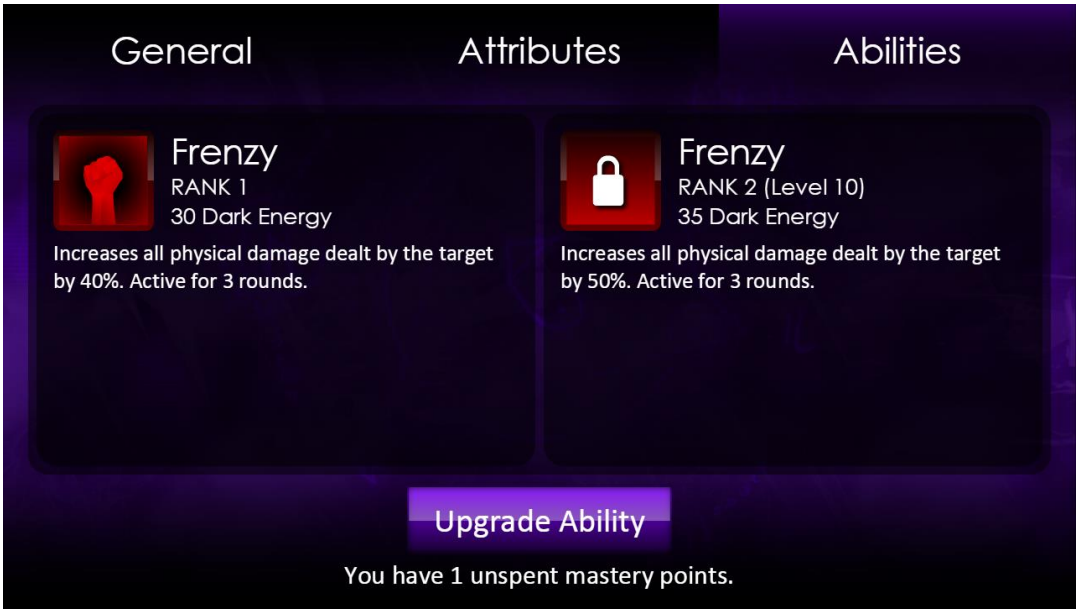
Abilities

Displays the ability sets known by the hero.



The upper panel contains the name of the currently selected set. Ability sets can be cycled through by tapping the adjacent arrow buttons.

The lower panel displays the abilities that belong to the selected set. By tapping one of them, the user is directed to a separate menu where it can see more details about that ability.



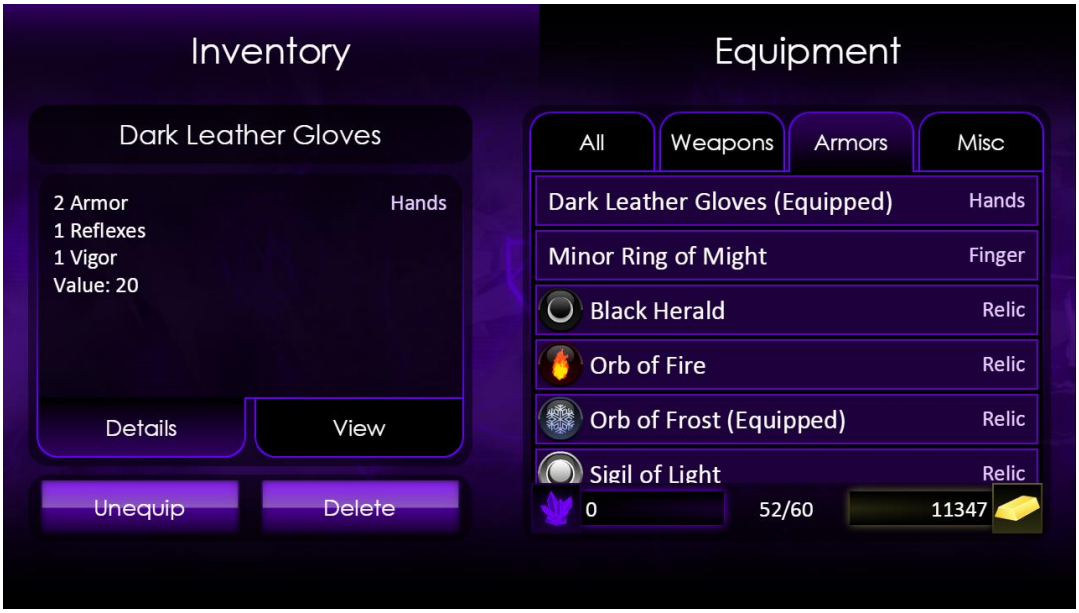
If all the requirements are met, the user can press **Upgrade Ability** to spend a mastery point and upgrade the current selection.

INVENTORY MENU

Allows the player to browse the inventory and view its equipment. It is divided into two categories.

Inventory

Displays the inventory and allows the player to use and delete items.



Equipment

Displays information about the items equipped by the player.

COMBAT

The combat screen appears when the player is engaged in battle.

The hero can perform an action by selecting a target and then pressing one of the abilities visible at the bottom of the screen. A player can change targets by simply tapping another unit. Information about the current and targeted units is displayed at the top of the screen.

INSTALLATION GUIDE

SYSTEM REQUIREMENTS

- A device running Windows Phone 8 or higher.
- 150 MB free space.
- 64 MB RAM.

RUNNING THE GAME

When Dark Energy will be released, the user will simply have to search for it on the Windows Phone Store and click on "Install" to download and run it. Until that point, however, the user must be a registered Windows Phone Developer and have the Windows Phone SDK[1] installed on its machine to be able to run the application.

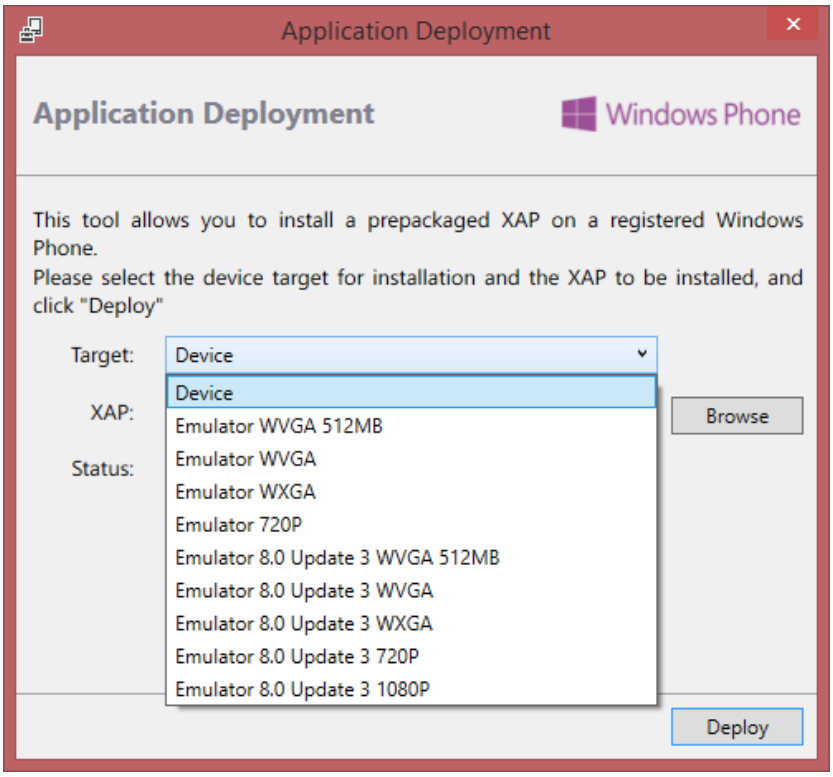
Deploying the project with the Application Deployment tool[2]

Once the SDK has been installed, launch the Application Deployment tool from the Start screen. You can find the Application Deployment tool by opening **All apps** view and locating it in the **Windows Phone SDK 8.0** group, or by typing **Application Deployment** from the Start screen to search for it.

You can also run the tool from the following location:

C:\Program Files (x86)\Microsoft SDKs\Windows Phone\v8.0\Tools\XAP Deployment\XapDeploy.exe

The application deployment tool starts, as shown below.



In the **Target** drop-down box, select either **Device** or one of the emulator options.

Click **Browse** and locate the x86 XAP file if you wish to deploy Dark Energy in an emulator or the ARM file if you want to run it on a device.

Click **Deploy**.

If the deployment is successful, the **Status** field displays **XAP Deployment Complete**.

ARCHITECTURE

VISION

The core architecture of this project is designed to be easy to maintain and highly efficient. It provides the developer with a high-performance game design framework that employs an intuitive programming interface centered on a variety of well-known design patterns.

TECHNOLOGY

Dark Energy is written in C# with XAML in Visual Studio 2013. The SharpDX[3] framework is used in order to access the power of DirectX technology. Data that loads at runtime is stored in XML files.

JUSTIFICATION

C#

C# was chosen due to its maintainability and object-oriented approach to programming. It is preferred over Java and Objective-C due to its elegant implementation of lambda functions, anonymous types and LINQ, the existence of properties, the power to overwrite operators and the fact that events are a built-in language feature.

A popular argument against using C# is that programs written in it are limited to operating systems developed by Microsoft. As the past few years have seen the rise of software that can compile C# code into native iOS and Android applications, this is no longer the case.

Reflection

Reflection is thoroughly used throughout the project. This provides a beneficial trade-off between computational speed and maintainability. Developer productivity and code simplicity is increased as objects can be loaded and instantiated at runtime by merely passing an ID through a method (see **DataManager**).

XML

Regarding content management, the decision to store data in the XML format as opposed to JSON was ultimately influenced by maintainability. XML files are easier to interpret by humans and the slightly larger space they occupy is computationally negligible.

Windows Phone

Lastly, from a financial perspective, targeting the Windows Phone market – one that currently lacks turn-based role-playing games of this particular kind and complexity, is a viable strategy to establish early popularity.

PROGRAMMING PATTERNS

Component-Oriented Programming

Classes formed of multiple domains without coupling them to one another. The **GameSystem** class exists to fulfill this purpose.

Factory Method
Many entities such as characters, items and abilities are created at runtime from external files by simply passing an ID to designated managers.

Observer
Event-driven programming. Most entities have listeners that let the framework know when their properties have altered.

State
Certain entities, such as characters, have a set of states that they can adopt. This enables an object to elegantly alter its behavior without resorting to massive conditional statements.

Type
Similar entities share the same behavior without having to designate any separate classes. For example, all enemies use a single class called **Enemy**.

BASE CLASSES

IGame
Nearly every class in the game implements the **IGame** interface. It contains the signatures of the fundamental methods and properties used throughout the project.

Type	Name	Description
Event	Loaded	Fires after the LoadContent method is called.
Property	IsLoaded	Becomes true after the LoadContent method is called.
Property	Visible	Toggles the visibility of the object.
Property	ColorIntensity	Controls the color intensity of the object.
Property	Opacity	Controls the opacity of the object.
Property	Rotation	Controls the rotation of the object.
Property	Parent	The parent object of the object.
Property	Scale	Controls the scale of the object.
Method	Initialize	Initializes the object. Is called before the LoadContent method.
Method	LoadContent	Loads all game assets belonging to the object.
Method	Update	Updates the object.
Method	Draw	Renders the object on the screen.
Method	UnloadContent	Unloads all game assets belonging to the object.

GameObject
The framework of Dark Energy consists of Game Objects, Managers and Game Systems. A **GameObject** implements the **IGame** interface and acts as the base class for every entity in the game. It defines an additional set of properties alongside the ones belonging to its interface.

Type	Name	Description
Event	PositionChanged	Fires after the X or Y property is changed.
Event	SizeChanged	Fires after the Scale property is changed.
Property	IsColorIntensityIndependent	Toggles whether the ColorIntensity property of the parent affects the object.

Property	IsOpacityIndependent	Toggles whether the Opacity property of the parent affects the object.
Property	IsRotationIndependent	Toggles whether the Rotation property of the parent affects the object.
Property	IsScaleIndependent	Toggles whether the Scale property of the parent affects the object.
Property	IsVisibilityIndependent	Toggles whether the Visibility property of the parent affects the object.
Property	HorizontalAlignment	Controls the horizontal alignment of the object.
Property	VerticalAlignment	Controls the vertical alignment of the object.
Property	X	Controls the X spatial coordinate of the object.
Property	Y	Controls the Y spatial coordinate of the object.
Property	PositionRectangle	Represents the portion of the screen that is occupied by the object.
Property	Offset	Controls the alignment offset of the object.
Property	Position	Controls the spatial coordinates of the object.
Property	Height	Gets the height of the object.
Property	Width	Gets the width of the object.
Property	Dimensions	Gets the dimensions of the object.

GameSystem

A **GameSystem** implements the **IGame** interface and is composed of multiple Game Objects. It defines an additional set of properties alongside the ones belonging to its interface.

Type	Name	Description
Event	SizeChanged	Fires after the Scale property is changed.
Property	IsColorIntensityIndependent	Toggles whether the ColorIntensity property of the parent affects the object.
Property	IsOpacityIndependent	Toggles whether the Opacity property of the parent affects the object.
Property	IsRotationIndependent	Toggles whether the Rotation property of the parent affects the object.
Property	IsScaleIndependent	Toggles whether the Scale property of the parent affects the object.
Property	IsVisibilityIndependent	Toggles whether the Visibility property of the parent affects the object.

TexturedElement

Extends the **GameObject** class and defines an additional set of properties. Behaves as a sprite[4].

Type	Name	Description
Property	Texture	Contains the texture resources of the object.
Property	Tapped	Indicates whether the object is tapped or not.
Property	AnimationRate	Controls the number of frames displayed per second.
Property	Frame	Controls the texture frame of the object.
Property	FrameCount	Controls the number of frames in a texture.
Property	Layer	Controls the texture layer of the object.
Property	TouchArea	Gets the tappable area of the object.
Property	TouchBoundaries	Extends the TouchArea.
Property	SpriteRectangle	Represents the portion of the texture that is currently visible.
Property	Path	The location of the texture.

Text

Extends the **GameObject** class and defines an additional set of properties. Font textures are created by the SharpDX framework when the project is compiled.

Type	Name	Description
Property	Font	Represents the font texture.
Property	String	Represents the text.
Property	Color	Represents the font color.
Property	Style	Represents the font style.
Property	MaxWidth	Controls the maximum width in pixels that the text can have.

MANAGING DATA

Dark Energy is the embodiment of a vast array of systems that work concomitantly with one another. A significant percentage of these systems handle complex data that is accessed at runtime. Textures are loaded by the **ContentManager**, courtesy of the SharpDX framework, while purely textual data that define objects such as characters, items, abilities etc. are loaded through the **DataManager** static class.

DataManager

A static class that aggressively uses reflection to load objects at runtime.

Type	Name	Description
Property	AbilityDirectory	Location of the ability directory.
Property	CharacterDirectory	Location of the character directory.
Property	ItemDirectory	Location of the item directory.
Property	RootDirectory	Location of the root directory.
Method	Load	Loads the object with the designated ID and type.

DataStorageManager

A static class that manages character progression data. Currently uses the System.IO.IsolatedStorage[5] library.

Type	Name	Description
Property	RootDirectory	Location of the root directory.
Method	Load	Loads the specified data.
Method	Save	Stores the specified data.
Method	Delete	Deletes the specified data.
Method	Flush	Writes all data to the disk.
Method	DeleteCharacter	Deletes the specified character.
Method	SaveCharacterLocation	Stores the location of the character. Due to its reflexive nature, no indexing is required.
Method	LoadScene	Loads the location of the character.

MANAGING SCENES

Every separate screen, such as a zone or a menu, is fundamentally a scene. When a user transitions between screens, the game unloads all assets and loads the ones required by the new scene. Dividing the game into scenes is thus an extremely cost-effective way of managing resources.

SceneManager

A static class that manages game systems which implement the **IScene** interface.

Type	Name	Description
Property	Current	The current scene.
Method	Play	Transitions to the specified scene.
Method	GoBack	Activates the OnBackKeyPress method of the current scene. Called when the user presses the back button.

CHARACTER SYSTEM

Character

Extends the **TexturedElement** class and defines an additional set of properties and methods.

Type	Name	Description
Property	Level	Controls the character level.
Property	Name	Controls the character name.
Property	AbilitySets	Controls the ability sets of the character.
Property	Base	Controls the base attributes of the character.
Property	Total	Controls the total attributes of the character.
Property	Health	Controls the character health.
Property	DarkEnergy	Controls the amount of Dark Energy that the character possesses.
Property	DefensiveElement	Controls the defensive element of the character.
Property	OffensiveElement	Controls the offensive element of the character.
Property	State	Controls the state of the character.
Property	Alive	Indicates whether the character is alive or not.
Property	ActiveEffects	Controls the effects that are currently affecting the character.
Method	AddEffect	Adds an effect on the character.

Hero

Extends the **Character** class and defines an additional set of properties. This unit is created by the player.

Type	Name	Description
Property	Features	Controls the character features.
Property	Equipment	Controls the character equipment.
Property	GenderBodyScale	Represents the effect of the character gender on its scale.
Property	IsHairVisible	Indicates whether the hair of the character is visible or not.

Enemy

Extends the **Character** class and defines an additional set of properties. Enemies are created at runtime from data stored in an external file.

Type	Name	Description
Property	Coins	Controls the amount of coins received by the hero when defeating the enemy.
Property	Experience	Controls the amount of experience received by the hero when defeating the enemy.

Method	ComputeAction	Computes an action that will be used by the enemy during its turn. Used in combat.

HeroSystem

Defines the hero and implements the **ILoadable** and **ISaveable** interfaces.

Type	Name	Description
Property	Level	Controls the hero level.
Property	Name	Controls the hero name.
Property	AttributePoints	Controls the attribute points that the hero possesses.
Property	MasteryPoints	Contorls the mastery points that the hero possesses.
Property	AbilitySets	Controls the ability sets of the hero.
Property	Base	Controls the base attributes of the hero.
Property	Total	Controls the total attributes of the hero.
Property	DefensiveElement	Controls the defensive element of the character.
Property	OffensiveElement	Controls the offensive element of the character.
Property	Features	Controls the hero features.
Property	Experience	Controls the hero experience.
Property	Equipment	Controls the hero equipment.
Method	IncreaseExperience	Increases the experience of the hero.

Ability

An ability constitutes a set of effects. This simplifies the process of adding new abilities as they are merely defined in an external file.

Type	Name	Description
Property	Id	Controls the ability ID.
Property	IconId	Controls the icon ID.
Property	Name	Controls the ability name.
Property	Rank	Controls the ability rank.
Property	HighestRank	Controls the highest rank that the ability can have.
Property	RequiredLevel	Controls the character level required to use this ability.
Property	DarkEnergyCost	Controls the amount of Dark Energy required to use the ability.
Property	Effects	Controls the effects that constitute the ability.
Property	TargetRestrictions	Controls the target restrictions.
Property	Animation	Controls the state adopted by the character while casting the ability.
Property	VisualId	Controls the ID of the visual effect seen while casting the ability.
Method	GetNextRank	Gets the next rank of the ability.
Method	GetPreviousRank	Gets the previous rank of the ability.
Method	IncreaseRank	Increases the ability rank.
Method	DecreaseRank	Decreases the ability rank.
Method	SetRank	Sets the ability rank to the specified value.
Method	GetDescription	Generates the ability description.

Effect

Effects are the building blocks of every ability.

Type	Name	Description
Property	Type	Controls the effect type.
Property	MinimumValue	Controls the minimum value of the effect.

Property	MaximumValue	Controls the maximum value of the effect.
Property	RoundsActive	Controls the number of rounds for which the effect is active.
Property	AreaEffect	Controls whether the effect targets an entire group or not.
Method	Apply	Applies the effect on the specified targets.

COMBAT SYSTEM

CombatSceneManager

A static class that manages the combat scene.

Type	Name	Description
Property	InBattle	Indicates whether the user is engaged in a battle or not.
Method	GetBattle	Gets the battle in which the user is engaged.
Method	Engage	Engages the user in the specified battle.
Method	ExitBattle	Removes the user from the battle.

Battle

Extends the **GameSystem** class and defines an additional set of events, properties and methods. It has been designed to be easily expanded in order to include online battles in the future.

Type	Name	Description
Event	RoundStarted	Fires after the BeginRound method is called.
Event	RoundEnded	Fires after the EndRound method is called.
Property	State	Controls the state of the battle.
Property	Victory	Indicates whether the user has won or not.
Property	Rounds	Indicates the number of rounds.
Property	AbilityManager	Controls the ability manager.
Property	AnimationManager	Controls the animation manager.
Property	AreaEffects	Controls the area effects.
Property	RewardManager	Controls the reward manager.
Property	TacticalMenu	Controls the tactical menu.
Property	Units	Controls the units.
Method	GetWorldScene	Gets the world scene.
Method	BeginRound	Starts the next round.
Method	EndRound	Ends the round.
Method	HideMenu	Hides the tactical menu.
Method	ShowMenu	Shows the tactical menu.

Units

Extends the **GameSystem** class and defines an additional set of events, properties and methods. Can contain up to six characters.

Type	Name	Description
Event	PositionChanged	Fires when the position of a unit is changed.
Event	CurrentChanged	Fires when the current unit is changed.
Event	TargetChanged	Fires when the targeted unit is changed.
Property	StartingPositions	Gets a list that contains the starting position of every unit.
Property	GroupA	Controls the friendly group.

Property	GroupB	Controls the enemy group.
Property	Actions	Controls the currently selected actions.
Property	CanSelectUnits	Controls whether the user can select a unit or not.
Property	Current	Controls the current unit.
Property	Target	Controls the targeted unit.
Property	CurrentIndex	Gets the index of the current unit.
Property	TargetIndex	Gets the index of the targeted unit.
Property	CurrentGroup	Gets the current group.
Property	TargetGroup	Gets the targeted group.
Property	All	Gets a list that contains every unit.
Property	Alive	Gets a list that contains every living unit.
Property	FirstAlive	Gets the first living unit from the specified group.
Property	CountAlive	Gets the number of living units from the specified group.
Method	GroupOf	Gets the group to which the specified unit belongs.
Method	InitializeSelections	Initializes unit selections.
Method	EnqueueAbility	Adds an ability to the queue.
Method	ComputeAttackOrder	Computes the order in which the selected actions will be executed.
Method	SelectNextCharacter	Selects the next unit.
Method	SelectPreviousCharacter	Selects the previous unit.
Method	ResetPositions	Resets the position of every unit.

TacticalMenu

Extends the **GameSystem** class and defines an additional set of properties. Represents the interface with which the user interacts during combat.

Type	Name	Description
Property	AbilityFrames	Controls the ability frames.
Property	UnitFrames	Controls the unit frames.

AbilityManager

Manages the abilities used during battle and controls the floating combat text.

Type	Name	Description
Property	IsCasting	Indicates whether the current unit is casting an ability or not.
Property	CastStarted	Indicates whether the current unit started casting an ability or not.
Property	CastCompleted	Indicates whether the current unit stopped casting an ability or not.
Property	CombatTextDisplayed	Indicates whether the combat text has been displayed or not.
Property	EffectsApplied	Indicates whether the effects of the ability have been applied or not.
Method	Next	Reinitializes all properties for the next round.
Method	Tick	Updates every active effect.
Method	Update	Updates its properties.

AnimationManager

Manages unit animations.

Type	Name	Description
Property	UnitAbilityAnimationStarted	Indicates whether the visual effect caused by the current ability started its animation or not.
Property	UnitAnimationEnded	Indicates whether the current unit finished its animation or not.
Method	Next	Reinitializes all properties for the next round.

Method	Update	Updates its properties.

RewardManager

Manages the rewards received by the user at the end of a battle.

Type	Name	Description
Method	GetReward	Generates the reward.
Method	Assign	Assigns the reward to the user.

INVENTORY SYSTEM

IItem

Contains the basic properties and methods used by every item.

Type	Name	Description
Property	Name	Represents the item name.
Property	Count	Represents the stack size of the item.
Property	StackLimit	Represents the number of items of this kind required to form a stack.
Property	Value	Represents the item value.
Property	IsStackable	Indicates whether the item is stackable or not.
Method	GetDescription	Generates a description for the item.

EquippableItem

Extends the **TexturedElement** class, implements the **IItem** interface and defines an additional set of properties.

Type	Name	Description
Property	Slot	Indicates the equipment slot that the item occupies.
Property	Attributes	Controls the attributes offered by the item.
Method	GetName	A static method that gets the name of the specified equipment slot.

Equipment

Extends the **GameSystem**, implements the **ILoadable** and **ISaveable** interfaces and defines an additional set of properties and methods.

Type	Name	Description
Property	Weapon	Controls item that occupies the weapon slot.
Property	Relic	Controls item that occupies the relic slot.
Property	Head	Controls item that occupies the head slot.
Property	Neck	Controls item that occupies the neck slot.
Property	Chest	Controls item that occupies the chest lost.
Property	Back	Controls item that occupies the back slot.
Property	Hands	Controls item that occupies the hands slot.
Property	Finger	Controls item that occupies the finger slot.
Property	Legs	Controls item that occupies the legs slot.
Property	Feet	Controls item that occupies the feet slot.
Property	Attributes	Gets the total attributes offered by the equipment.
Method	ToList	Converts the class to a list of equippable items.

Method	GetIdList	Gets a list that contains the ID of every item.
Method	Clear	Unequips every item.
Method	Unequip	Unequips the item that occupies the specified slot.
Method	SetState	Changes the state of every item based on the hero.
Method	StartDrawing	Draws the items that underlap the hero.
Method	FinishDrawing	Draws the items that overlap the hero.

InventorySystem

Implements the **ILoadable** and **ISaveable** interfaces and defines an additional set of properties and methods.

Type	Name	Description
Property	Capacity	Controls the number of items that can be possessed by the hero.
Property	Coins	Controls the coins possessed by the hero.
Property	DarkCrystals	Controls the dark crystals possessed by the hero.
Property	Items	Controls the items in the inventory.
Property	Count	Represents the number of items in the inventory.
Property	Equipment	Controls the equipment of the hero.
Method	Contains	Indicates whether the inventory contains the specified item or not.
Method	IsUnique	Indicates whether an item of the specified ID and Count is found exactly once in the inventory or not.
Method	IsUniqueStack	Indicates whether an item of the specified ID is found exactly once in the inventory or not.
Method	IsEquipped	Indicates whether the specified item is equipped or not.
Method	GetItemWithId	Gets the item with the specified ID.
Method	GetId	Gets the ID of the specified item.
Method	Equip	Equips the specified target with the specified items.
Method	Clear	Resets the inventory.
Method	Add	Adds the specified item to the inventory.
Method	Remove	Removes the specified item from the inventory.

REFERENCES

1. Windows Phone SDK
   https://dev.windowsphone.com/en-us/downloadsdk

2. How to deploy and run an app for Windows Phone 8
   http://msdn.microsoft.com/en-us/library/windowsphone/develop/ff402565(v=vs.105).aspx

3. SharpDX Framework
   http://sharpdx.org/download/

4. Sprite
   http://en.wikipedia.org/wiki/Sprite_(computer_graphics)

5. System.IO.IsolatedStorage
   http://msdn.microsoft.com/en-us/library/system.io.isolatedstorage.isolatedstoragefile.aspx