CS499
SENIOR CAPSTONE PROJECT

Dungeon Warrior

Software Design Document

Version 2.0

Randall Rowland

24 September 2016

REVISION HISTORY

Date	Version	Description	Author
29 June 2016	1.0	Initial version of Dungeon Warrior's Software Design Document	Randall Rowland
18 July 2016	1.1	Added interface graphic	Randall Rowland
2 August 2016	1.2	Added flow chart graphic and interface operations.	Randall Rowland
23 August 2016	1.3	Added UML and citations	Randall Rowland
23 August 2016	1.4	Swapped Flow Charts	Randall Rowland
24 September 2016	2.0	Doxygen Class Documentation	Randall Rowland

CONTENTS

1. INTRODUCTION

The Software Design Document is a document to provide documentation which will be used to aid in software development by providing the details for how the software should be built. Within the Software Design Document are narrative and graphical documentation of the software design for the project including sequence diagrams, collaboration models, object behavior models, and other supporting requirement information.

1.1. PURPOSE

The purpose of this software design document is to provide a low-level description of Dungeon Warrior, providing insight into the structure and design of each component. Topics covered include the following:

- Class hierarchies and interactions
- Data flow and design
- Processing narratives
- Algorithmic models
- Design constraints and restrictions
- User interface design
- Test cases and expected results

In short, this document is meant to equip the reader with a solid understanding of the inner workings of Dungeon Warrior.

1.2. GOALS AND OBJECTIVES

Dungeon Warrior is a single, comprehensive programming project using the C++ programming language. The goal is a proof of concept program to demonstrate learning objectives learned throughout American Sentinel University's Bachelor of Science in Computer Science Game Programming Specialization degree program. Dungeon Warrior will incorporate the learning objectives from:

- CS130 – Introduction to Computer Programming
- CS205 – Intermediate Computer Programming
- CS221 – Software Engineering
- GP210 – Introduction to Game Design
- GP221 – Introduction to Game Programming
- GP312 – Computer Graphics Programming
- GP435 – Artificial Intelligence for Gaming

The objectives of Dungeon Warrior:

- Apply software engineering techniques to a larger-scale problem
- Integrate appropriate computer science theory, concepts, and methods
- Demonstrate proper documentation
- Display comprehensive programming knowledge

1.3. DEFINITIONS, ACRONYMS, AND ABBREVIATIONS

- **ADT –** Abstract Data Type. A collection of data values together with a set of well-specified operations on that data.
- **AI** – Artificial Intelligence.
- **API** – Application Program Interface
- **Collision** – Determining if an object has intersected another object or overlapped relevant background scenery.
- **DirectX** - a collection of APIs for handling tasks related to multimedia, especially game programming and video, on Microsoft platforms.
- **Object** – Is a data structure that has state (data) and behavior (code). Objects correspond to things found in the real world.
- **OOP** – Object Oriented Programming. Programming language model organized around objects rather than "actions" and data rather than logic.
- **OpenGL** - Open Graphics Library (OpenGL) is a cross-language, cross-platform API for rendering 2D and 3D vector graphics.
- **Scholarship** – Academic study or achievement; learning of a high level
- **SDD** – Software Design Document.
- **UML** – Unified Modeling Language. For definition and uses, see http://www.uml.org/what-is-uml.htm

1.4. REFERENCES

Christopho. (2016, January 25). Zelda ALTTP resource pack for Solarus. Retrieved June 29, 2016, from https://github.com/christopho/solarus-alttp-pack

Customan. (2012, November 20). Folder Games Icon. Retrieved July 18, 2016, from http://www.iconarchive.com/show/christmas-icons-by-custo-man/Folder-Games-icon.html

Morrison, M. (2005). Beginning game programming. Indianapolis, IN: SAMS.

Schell, J. (2008). *The art of game design: A book of lenses*. Amsterdam: Elsevier/Morgan Kaufmann.

1.5. LICENSE

1.5.1. SOFTWARE DESIGN DOCUMENT/SOURCE CODE LICENSE

This document, the icon and the source code for Dungeon Warrior are licensed under a Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License.

Under this license, you are free to:

- **Share** – copy and redistribute the material in any medium or format
- **Adapt** – remix, transform, and build upon the material

Under the following terms:

- **Attribution** – You must give appropriate credit, provide a link to the license, and indicate if changes were made. You may do so in any reasonable manner, but not in any way that suggest the licensor endorses you or your use.
- **NonCommercial** – You may not use the material for commercial purposes.
- **ShareAlike** – If you remix, transform, or build upon the material, you must distribute your contribution under the same license as the original.

1.5.2. ZELDA: A LINK TO THE PAST COPYRIGHT

The Zelda fonts, music, sounds, sprites, tile sets, and other resources used from Zelda: A Link to the Past are copyrighted and owned by Nintendo.

As Dungeon Warrior is for scholarship, these resources fall under Fair Use. For this particular instance. However, if the project is shared outside of educational purposes or for commercial use, these resources must be removed.

17 U.S. CODE § 107 - LIMITATIONS ON EXCLUSIVE RIGHTS: FAIR USE

Notwithstanding the provisions of sections 106 and 106A, the fair use of a copyrighted work, including such use by reproduction in copies or phonorecords or by any other means specified by that section, for purposes such as criticism, comment, news reporting, teaching (including multiple copies for classroom use), scholarship, or research, is not an infringement of copyright. In determining whether the use made of a work in any particular case is a fair use the factors to be considered shall include—

(1) the purpose and character of the use, including whether such use is of a commercial nature or is for nonprofit educational purposes;

(2) the nature of the copyrighted work;

(3) the amount and substantiality of the portion used in relation to the copyrighted work as a whole; and

(4) the effect of the use upon the potential market for or value of the copyrighted work.

The fact that a work is unpublished shall not itself bar a finding of fair use if such finding is made upon consideration of all the above factors.

2. DESIGN OVERVIEW

2.1. INTRODUCTION

The Design Overview is section to introduce and give a brief overview of the design. The System Architecture is a way to give the overall view of a system and to place it into context with external systems. This allows for the reader and user of the document to orient themselves to the design and see a summary before proceeding into the details of the design.

2.2. TECHNOLOGIES USED

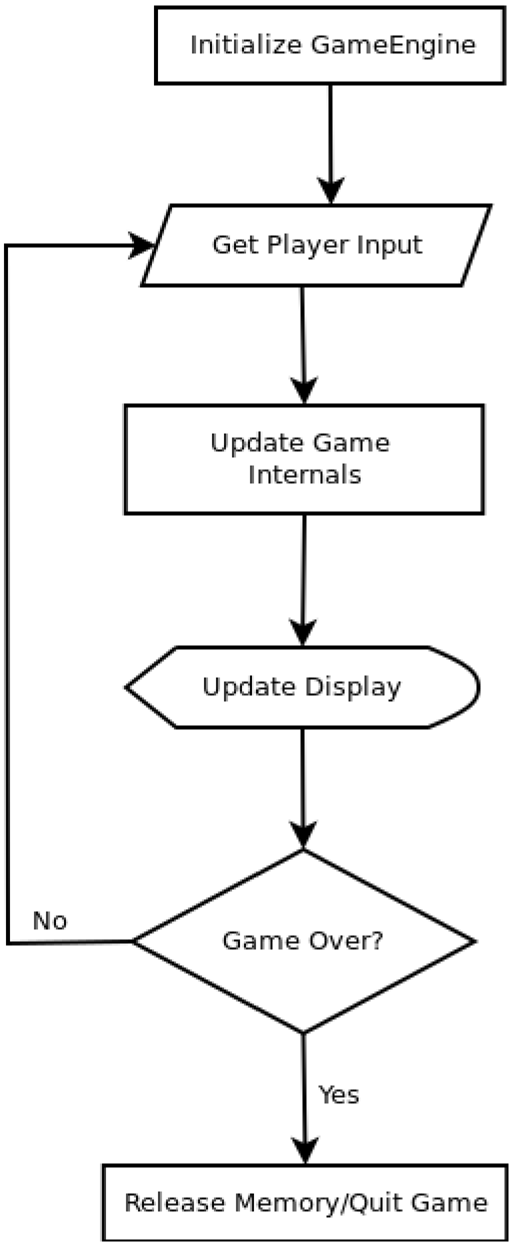
2.2.1. HARDWARE

- Hewlett-Packard ProBook 640G1
  - Intel® Core™ i3-4000M
  - 8 GB Ram
  - Windows 7 Enterprise SP1 (64-bit)
- Lenovo X1 Carbon
  - Intel® Core™ i7-5667U
  - 8 GB Ram
  - Ubuntu 16.04 LTS (64-bit)
- Alienware Mx11-R2
  - Intel® Core™ i7
  - 8 GB Ram
  - Windows 7 Home Edition SP1 (64-bit)
- Custom Desktop PC
  - AMD Phenom
  - 16 GB Ram
  - Ubuntu 16.04 LTS (64-bit)

2.2.2. SOFTWARE

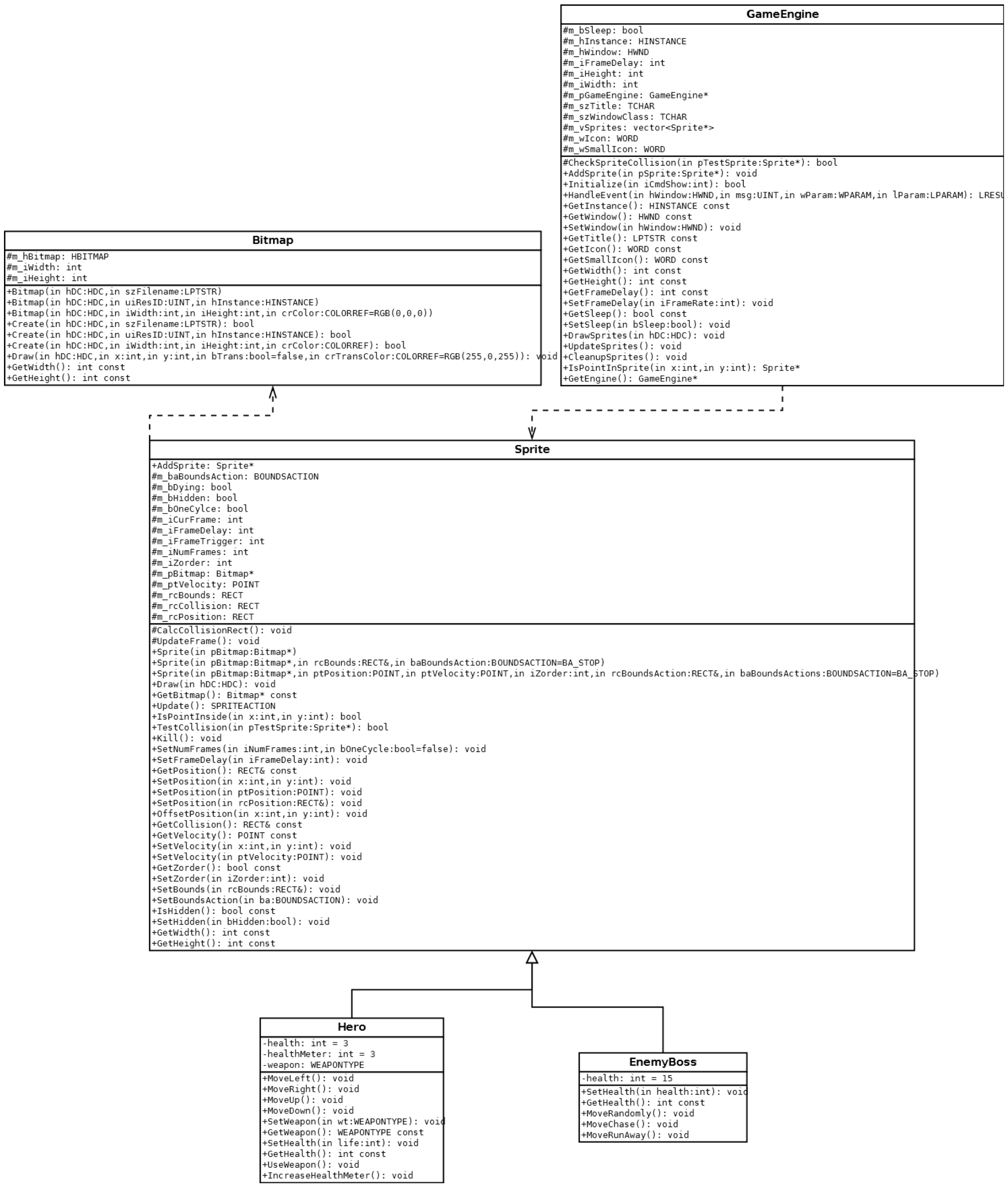
- Microsoft Visual Studio Community 2015 (Version 14.0.25421.03 Update 3)
- Microsoft Visual Studio Code
- Microsoft Office 2016 Professional Plus
- Atlassian SourceTree
- Pinta 1.6
- Microsoft Word 2013
- Syntevo GmbH SmartGit 7.1
- Dia 0.97.3
- Geany 1.27
- GitKraken
- Doxygen 1.8.11

2.3. SYSTEM ARCHITECTURE

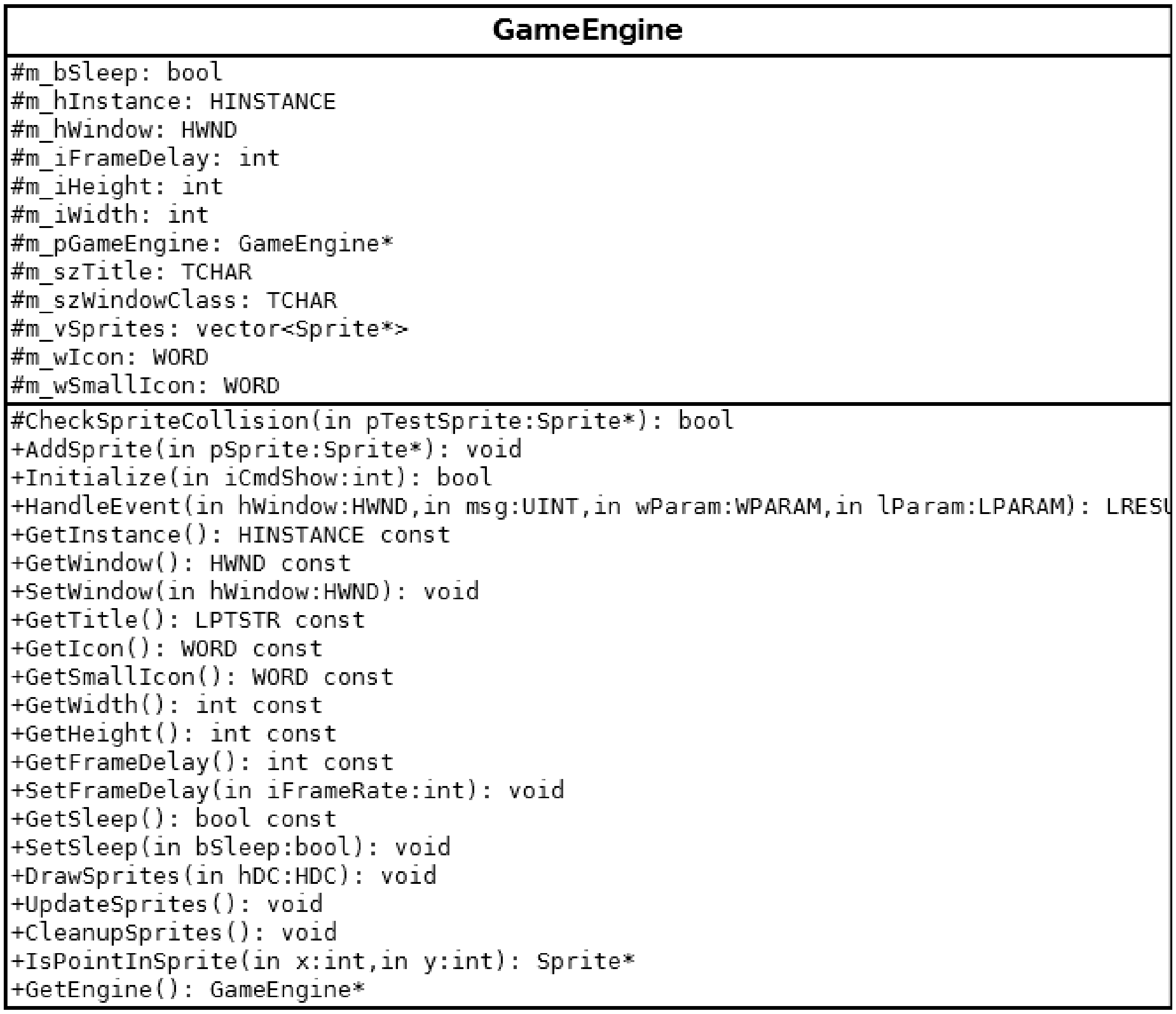


2.4. SYSTEM INTERFACES AND OPERATION

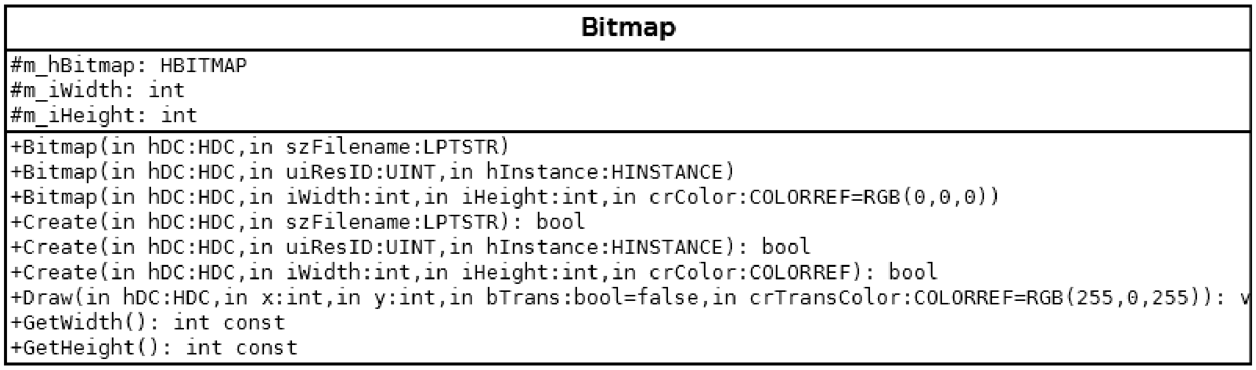
Overview of how the system interacts with one another.


GameEngine
#m_bSleep: bool
#m_hInstance: HINSTANCE
#m_hWindow: HWND
#m_iFrameDelay: int
#m_iHeight: int
#m_iWidth: int
#m_pGameEngine: GameEngine*
#m_szTitle: TCHAR
#m_szWindowClass: TCHAR
#m_vSprites: vector<Sprite*>
#m_wIcon: WORD
#m_wSmallIcon: WORD
#CheckSpriteCollision(in pTestSprite:Sprite*): bool
+AddSprite(in pSprite:Sprite*): void
+Initialize(in iCmdShow:int): bool
+HandleEvent(in hWindow:HWND,in msg:UINT,in wParam:WPARAM,in lParam:LPARAM): LRESU
+GetInstance(): HINSTANCE const
+GetWindow(): HWND const
+SetWindow(in hWindow:HWND): void
+GetTitle(): LPTSTR const
+GetIcon(): WORD const
+GetSmallIcon(): WORD const
+GetWidth(): int const
+GetHeight(): int const
+GetFrameDelay(): int const
+SetFrameDelay(in iFrameRate:int): void
+GetSleep(): bool const
+SetSleep(in bSleep:bool): void
+DrawSprites(in hDC:HDC): void
+UpdateSprites(): void
+CleanupSprites(): void
+IsPointInSprite(in x:int,in y:int): Sprite*
+GetEngine(): GameEngine*
Bitmap
#m_hBitmap: HBITMAP
#m_iWidth: int
#m_iHeight: int
+Bitmap(in hDC:HDC,in szFilename:LPTSTR)
+Bitmap(in hDC:HDC,in uiResID:UINT,in hInstance:HINSTANCE)
+Bitmap(in hDC:HDC,in iWidth:int,in iHeight:int,in crColor:COLORREF=RGB(0,0,0))
+Create(in hDC:HDC,in szFilename:LPTSTR): bool
+Create(in hDC:HDC,in uiResID:UINT,in hInstance:HINSTANCE): bool
+Create(in hDC:HDC,in iWidth:int,in iHeight:int,in crColor:COLORREF): bool
+Draw(in hDC:HDC,in x:int,in y:int,in bTrans:bool=false,in crTransColor:COLORREF=RGB(255,0,255)): void
+GetWidth(): int const
+GetHeight(): int const
Sprite
+AddSprite: Sprite*
#m_baBoundsAction: BOUNDSACTION
#m_bDying: bool
#m_bHidden: bool
#m_bOneCylce: bool
#m_iCurFrame: int
#m_iFrameDelay: int
#m_iFrameTrigger: int
#m_iNumFrames: int
#m_iZorder: int
#m_pBitmap: Bitmap*
#m_ptVelocity: POINT
#m_rcBounds: RECT
#m_rcCollision: RECT
#m_rcPosition: RECT
#CalcCollisionRect(): void
#UpdateFrame(): void
+Sprite(in pBitmap:Bitmap*)
+Sprite(in pBitmap:Bitmap*,in rcBounds:RECT&,in baBoundsAction:BOUNDSACTION=BA_STOP)
+Sprite(in pBitmap:Bitmap*,in ptPosition:POINT,in ptVelocity:POINT,in iZorder:int,in rcBoundsAction:RECT&,in baBoundsActions:BOUNDSACTION=BA_STOP)
+Draw(in hDC:HDC): void
+GetBitmap(): Bitmap* const
+Update(): SPRITEACTION
+IsPointInside(in x:int,in y:int): bool
+TestCollision(in pTestSprite:Sprite*): bool
+Kill(): void
+SetNumFrames(in iNumFrames:int,in bOneCycle:bool=false): void
+SetFrameDelay(in iFrameDelay:int): void
+GetPosition(): RECT& const
+SetPosition(in x:int,in y:int): void
+SetPosition(in ptPosition:POINT): void
+SetPosition(in rcPosition:RECT&): void
+OffsetPosition(in x:int,in y:int): void
+GetCollision(): RECT& const
+GetVelocity(): POINT const
+SetVelocity(in x:int,in y:int): void
+SetVelocity(in ptVelocity:POINT): void
+GetZorder(): bool const
+SetZorder(in iZorder:int): void
+SetBounds(in rcBounds:RECT&): void
+SetBoundsAction(in ba:BOUNDSACTION): void
+IsHidden(): bool const
+SetHidden(in bHidden:bool): void
+GetWidth(): int const
+GetHeight(): int const
Hero
-health: int = 3
-healthMeter: int = 3
-weapon: WEAPONTYPE
+MoveLeft(): void
+MoveRight(): void
+MoveUp(): void
+MoveDown(): void
+SetWeapon(in wt:WEAPONTYPE): void
+GetWeapon(): WEAPONTYPE const
+SetHealth(in life:int): void
+GetHealth(): int const
+UseWeapon(): void
+IncreaseHealthMeter(): void
EnemyBoss
-health: int = 15
+SetHealth(in health:int): void
+GetHealth(): int const
+MoveRandomly(): void
+MoveChase(): void
+MoveRunAway(): void

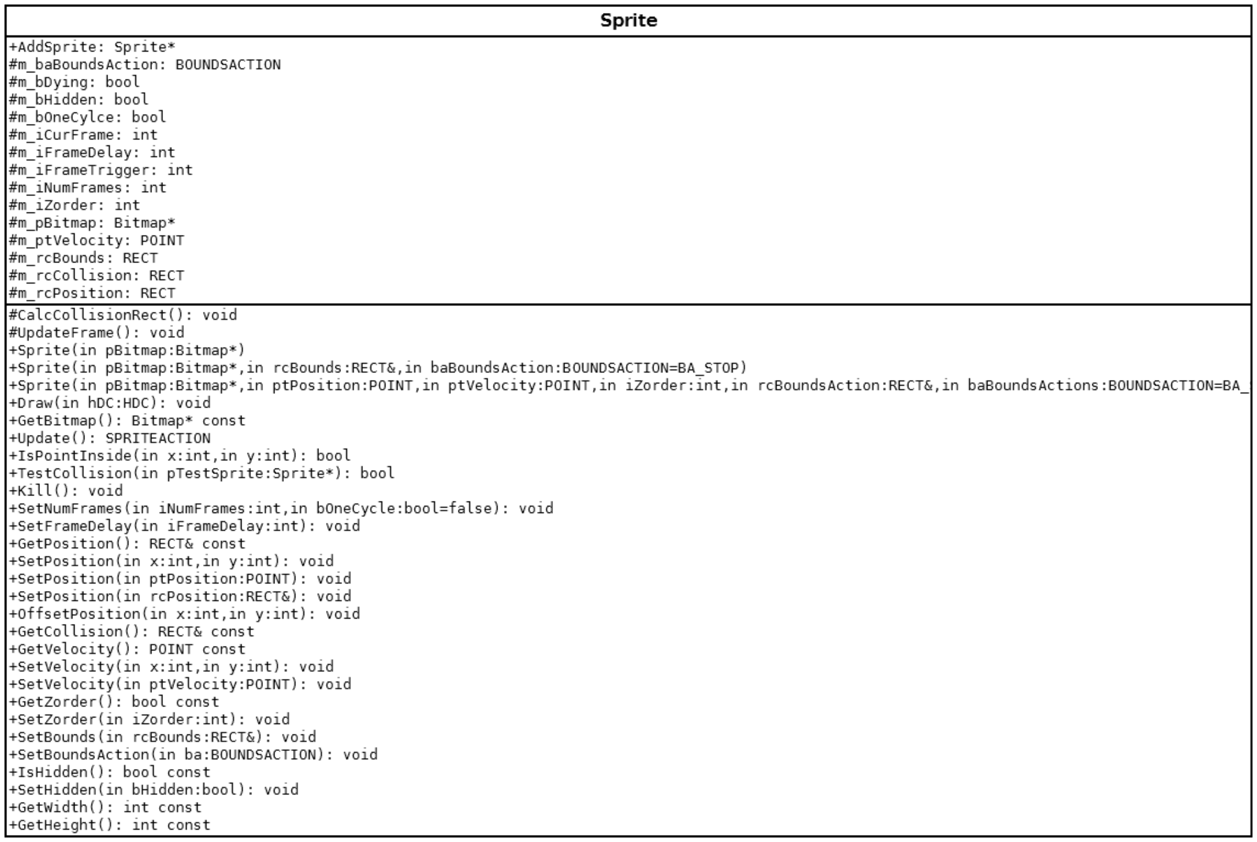
The following is a close up of each class of UML:



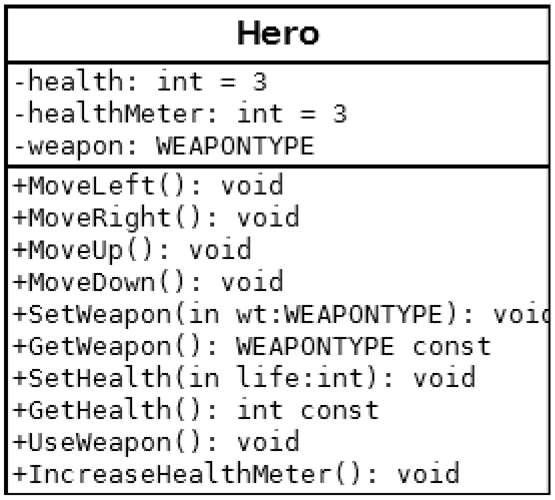
```
GameEngine
#m_bSleep: bool
#m_hInstance: HINSTANCE
#m_hWindow: HWND
#m_iFrameDelay: int
#m_iHeight: int
#m_iWidth: int
#m_pGameEngine: GameEngine*
#m_szTitle: TCHAR
#m_szWindowClass: TCHAR
#m_vSprites: vector<Sprite*>
#m_wIcon: WORD
#m_wSmallIcon: WORD
#CheckSpriteCollision(in pTestSprite:Sprite*): bool
+AddSprite(in pSprite:Sprite*): void
+Initialize(in iCmdShow:int): bool
+HandleEvent(in hWindow:HWND,in msg:UINT,in wParam:WPARAM,in lParam:LPARAM): LRESU
+GetInstance(): HINSTANCE const
+GetWindow(): HWND const
+SetWindow(in hWindow:HWND): void
+GetTitle(): LPTSTR const
+GetIcon(): WORD const
+GetSmallIcon(): WORD const
+GetWidth(): int const
+GetHeight(): int const
+GetFrameDelay(): int const
+SetFrameDelay(in iFrameRate:int): void
+GetSleep(): bool const
+SetSleep(in bSleep:bool): void
+DrawSprites(in hDC:HDC): void
+UpdateSprites(): void
+CleanupSprites(): void
+IsPointInSprite(in x:int,in y:int): Sprite*
+GetEngine(): GameEngine*
```



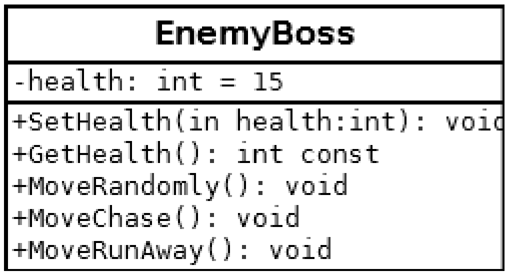
```
Bitmap
#m_hBitmap: HBITMAP
#m_iWidth: int
#m_iHeight: int
+Bitmap(in hDC:HDC,in szFilename:LPTSTR)
+Bitmap(in hDC:HDC,in uiResID:UINT,in hInstance:HINSTANCE)
+Bitmap(in hDC:HDC,in iWidth:int,in iHeight:int,in crColor:COLORREF=RGB(0,0,0))
+Create(in hDC:HDC,in szFilename:LPTSTR): bool
+Create(in hDC:HDC,in uiResID:UINT,in hInstance:HINSTANCE): bool
+Create(in hDC:HDC,in iWidth:int,in iHeight:int,in crColor:COLORREF): bool
+Draw(in hDC:HDC,in x:int,in y:int,in bTrans:bool=false,in crTransColor:COLORREF=RGB(255,0,255)): v
+GetWidth(): int const
+GetHeight(): int const
```



```
Sprite
+AddSprite: Sprite*
#m_baBoundsAction: BOUNDSACTION
#m_bDying: bool
#m_bHidden: bool
#m_bOneCylce: bool
#m_iCurFrame: int
#m_iFrameDelay: int
#m_iFrameTrigger: int
#m_iNumFrames: int
#m_iZorder: int
#m_pBitmap: Bitmap*
#m_ptVelocity: POINT
#m_rcBounds: RECT
#m_rcCollision: RECT
#m_rcPosition: RECT
#CalcCollisionRect(): void
#UpdateFrame(): void
+Sprite(in pBitmap:Bitmap*)
+Sprite(in pBitmap:Bitmap*,in rcBounds:RECT&,in baBoundsAction:BOUNDSACTION=BA_STOP)
+Sprite(in pBitmap:Bitmap*,in ptPosition:POINT,in ptVelocity:POINT,in iZorder:int,in rcBoundsAction:RECT&,in baBoundsActions:BOUNDSACTION=BA_
+Draw(in hDC:HDC): void
+GetBitmap(): Bitmap* const
+Update(): SPRITEACTION
+IsPointInside(in x:int,in y:int): bool
+TestCollision(in pTestSprite:Sprite*): bool
+Kill(): void
+SetNumFrames(in iNumFrames:int,in bOneCycle:bool=false): void
+SetFrameDelay(in iFrameDelay:int): void
+GetPosition(): RECT& const
+SetPosition(in x:int,in y:int): void
+SetPosition(in ptPosition:POINT): void
+SetPosition(in rcPosition:RECT&): void
+OffsetPosition(in x:int,in y:int): void
+GetCollision(): RECT& const
+GetVelocity(): POINT const
+SetVelocity(in x:int,in y:int): void
+SetVelocity(in ptVelocity:POINT): void
+GetZorder(): bool const
+SetZorder(in iZorder:int): void
+SetBounds(in rcBounds:RECT&): void
+SetBoundsAction(in ba:BOUNDSACTION): void
+IsHidden(): bool const
+SetHidden(in bHidden:bool): void
+GetWidth(): int const
+GetHeight(): int const
```



```
Hero
-health: int = 3
-healthMeter: int = 3
-weapon: WEAPONTYPE
+MoveLeft(): void
+MoveRight(): void
+MoveUp(): void
+MoveDown(): void
+SetWeapon(in wt:WEAPONTYPE): voi
+GetWeapon(): WEAPONTYPE const
+SetHealth(in life:int): void
+GetHealth(): int const
+UseWeapon(): void
+IncreaseHealthMeter(): void
```



```
EnemyBoss
-health: int = 15
+SetHealth(in health:int): voi
+GetHealth(): int const
+MoveRandomly(): void
+MoveChase(): void
+MoveRunAway(): void
```

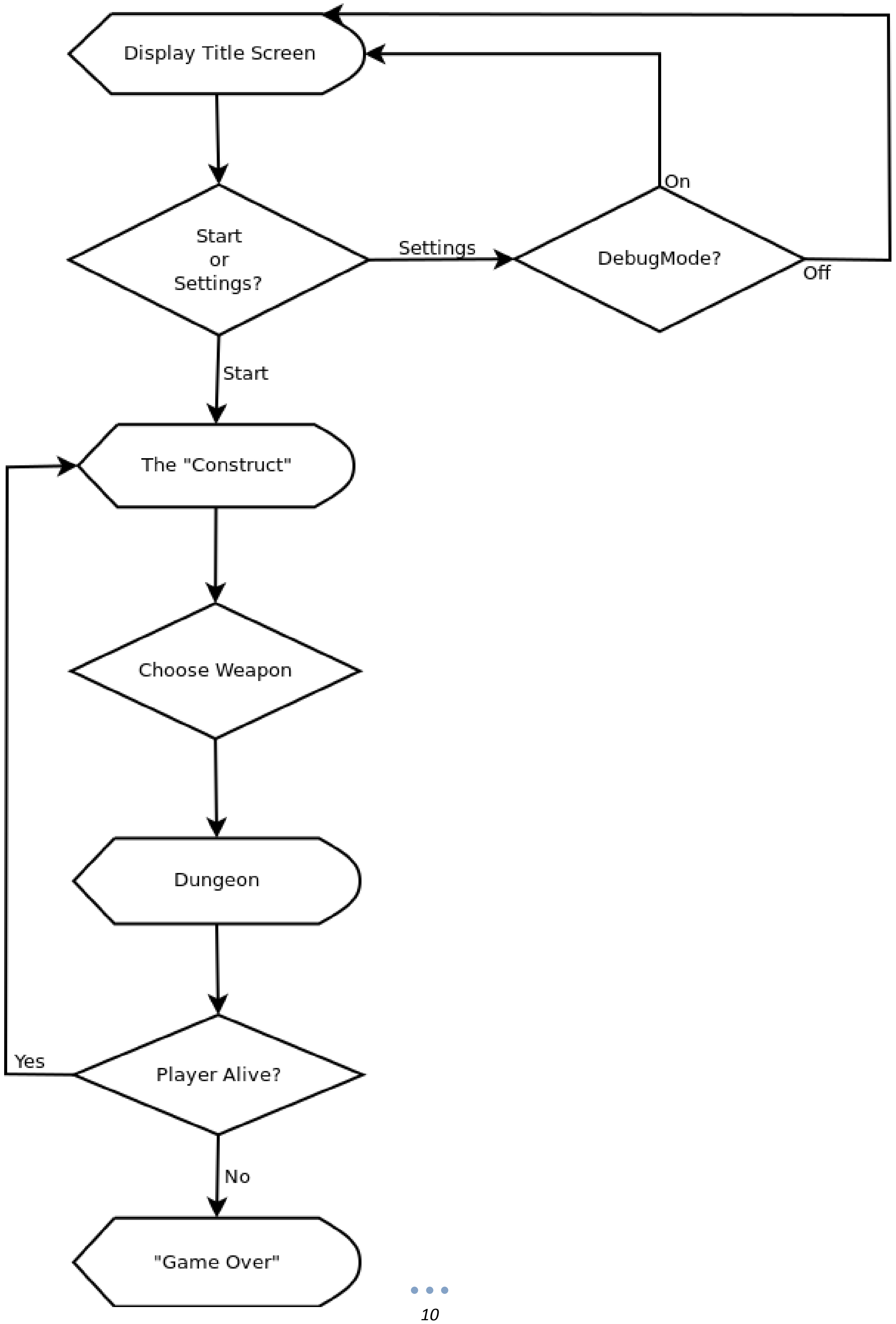
2.5. CONSTRAINTS AND ASSUMPTIONS

Due to the learning curve and ever changing technology of Microsoft's DirectX API and OpenGL API, this game was programmed using just Microsoft's Windows API. Although, DirectX or OpenGL would provide a more robust game, the eight-week time constraint on the project makes using those APIs near impossible. I would also have to test it out on more machines and see which dependencies would have to be included to ensure it runs. The assumption is that by using the Windows API, this game should run on any modern Windows version.

3. USER INTERFACE DESIGN

3.1. DESCRIPTION OF THE USER INTERFACE

Title Screen. The will show a beautiful graphical background with text over the top show casing the title of the game, "Dungeon Warrior". Towards the bottom of the title screen will be two items that will be selectable by using the arrow keys to highlight one of the items and using 'Enter' or 'Space Bar' to select it. The two items will be graphical text that states: "Start Game" or "Settings".
The "Construct". This is where the game will start off at. The name comes from The Matrix. In The Matrix, the Construct is a virtual workspace that is used as their loading program. They are able to load virtual objects that are then used within The Matrix. This same concept applies here. When the game starts off you will be in an empty room where an old man will greet you. If you know about The Hero's Journey concept, this old man is the Mentor. He will tell you to choose one of the three weapons in front of you, which will be a bow with arrows, a sword, and a boomerang. Once you select one and confirm your selection, you will be transported to the dungeon. The arrow keys will move the player around and the 'Spacebar' will be used to make selections. "This…is the Construct. It's our loading program. We can load anything, from clothing…to equipment...weapons…training simulations…anything we need." -Morpheus
Dungeon. Inside the dungeon is where the player will battle enemies using the weapon they chose from The Construct. Once all the enemies are defeated they will be transported back to The Construct to switch weapons if they choose to. Each time the player defeats all the enemies and returns from The Construct, the dungeon will increase difficulty. More enemies or a completely different enemy or a combination of both will happen each time the play enters the dungeon. If the player is unable to defeat the enemies and dies, the game ends and will display the game over screen. The player will use the arrow keys and 'Spacebar' on the keyboard to control their avatar. The arrow keys will move the player around and the 'Spacebar' will activate the avatar's weapon.
Game Over. If the player is unable to defeat the enemies in the dungeon and dies. The game over screen will appear to let the player know how many waves of enemy they were able to defeat. From this screen they can press 'Enter' or 'Spacebar' and the game will release resources from memory and close the program.
Settings. When settings are selected from the title screen, a pop-up balloon style window will appear. At the top of the window will be text that says, 'DEBUG'. Below that will be text that will say 'ON' and 'OFF'. Default when the program starts will be set to off. No matter which one is selected, the window will close and you will be presented with the Title Screen again. If debug is set to on, the frames per second will be displayed in the corner and boundary boxes around sprites will display so you can see collisions. The player will be able to select on or off using the arrow keys and confirming the selection with the spacebar.

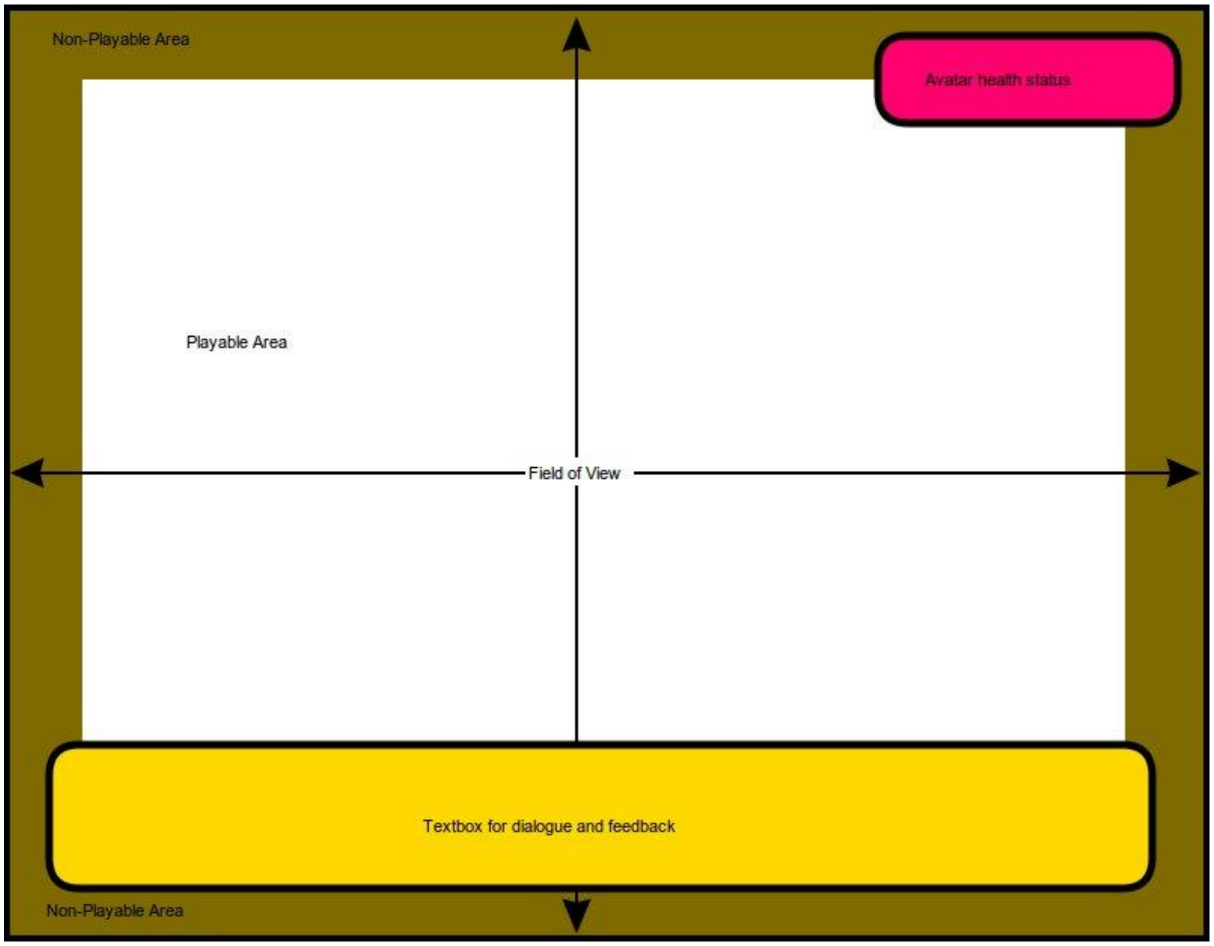


Display Title Screen
Start or Settings?
Settings
DebugMode?
On
Off
Start
The "Construct"
Choose Weapon
Dungeon
Player Alive?
Yes
No
"Game Over"

3.2. INTERFACE DESIGN RULES

The interface design rules for Dungeon Warrior are derived from The Art of Game Design. I used the Lens of Simplicity and Transcendence, Lens of The Hero's Journey, Lens of Status, Lens of Action, Lens of Goals, Lens of Virtual Interface, and Lens of the Avatar.

3.3. OBJECTS AND ACTIONS



This is a general layout of how the user interface will look. Proportions may not be exact in the final product. End user will interact with the user interface and avatar using the arrow keys and space bar on the keyboard. **Field of View:** Inside of the Field of View will be a set size and the end user will not be able to change the size to ensure correct aspect ratio and view. Field of View will be contained within the Window and will not scroll. **Non-Playable Area:** Inside of that window will be a border. Border will use wall type graphic to give the illusion the avatar is in a room. This is depicted above with the brown/bronze color and labeled as a Non-Playable Area. This will stop the avatar from "walking" off the screen. Although depicted all the way to the edge of the Field of View, this is only an example. Some levels may be smaller and have a smaller Playable Area. **Playable Area:** The avatar will be able to move anywhere within the white area or Playable area. Border will prevent avatar from accessing non-playable areas. EXCEPTION: Elements may be added to the playable area to give an aesthetic look and challenge the avatar. The avatar may or may not be able to "walk" through those elements.

Textbox: This portion depicted with the yellow/gold box above will not be visible all the time. This will be a popup box when the avatar is interacting with other entities. When this popup box appears, all elements within the Playable Area will pause. The main focus will be on the textbox and the end user will only be able to interact with the textbox until it is complete and goes away. Then the Playable Area will resume normal game play.
Avatar Health Box: The pink box depicted in the top right corner will be sometime of floating health meter to provide the avatars health status to the end user. This will be visible at all times during game play.

4. CLASS DOCUMENTATION

4.1. BITMAP CLASS REFERENCE

Takes a bitmap and turns it into a Bitmap object that will be drawn to the screen.

#include <Bitmap.h>

PUBLIC MEMBER FUNCTIONS

- Bitmap ()

  Empty contructor that sets width and height to 0, and handle to NULL.
- Bitmap (HDC, LPTSTR)
- Bitmap (HDC, UINT, HINSTANCE)
- Bitmap (HDC, int, int, COLORREF crColor=RGB(0, 0, 0))
- virtual ~Bitmap ()

  A destructor that calls the Free() helper fuction to delete the object.
- bool Create (HDC, LPTSTR)
- bool Create (HDC, UINT, HINSTANCE)
- bool Create (HDC, int, int, COLORREF)
- void Draw (HDC, int, int, bool bTrans=false, COLORREF crTransColor=RGB(255, 0, 255))
- int GetWidth () const

  Returns the width of the bitmap.
- int GetHeight () const

  Returns the height of the bitmap.
- void DrawPart (HDC, int, int, int, int, int, int, BOOL, COLORREF)

PROTECTED MEMBER FUNCTIONS

- void Free ()

  A helper fuction that deletes the Bitmap object and releases the handle from memory.

PROTECTED ATTRIBUTES

- HBITMAP m_hBitmap

The handle to the bitmap and its copy of image bits.

- int m_iWidth

The width in pixels of the bitmap.

- int m_iHeight

The height in pixels of the bitmpa.

4.1.1DETAILED DESCRIPTION

Takes a bitmap and turns it into a Bitmap object that will be drawn to the screen.

There are four different types of constructors used to create a Bitmap object. You can draw the bitmap with or without transparency. Also allows only a part of a Bitmap to be drawn. This can be use full if you have 'frames' that could allow animation or if you want to just use a tilesheet for several images.

4.1.2CONSTRUCTOR & DESTRUCTOR DOCUMENTATION

4.1.2.1 BITMAP::BITMAP (HDC *HDC,* LPTSTR *SZFILENAME*)

Constructor that

PARAMETERS

hDC	The handle to the screen context device
szFileName	The file of the bitmap the Bitmap object will be created from

4.1.2.2 BITMAP::BITMAP (HDC *HDC,* UINT *UIRESID,* HINSTANCE *HINSTANCE*)

Constructor that

PARAMETERS

hDC	The handle to the screen context device
uiResID	The ID of the bitmap from the resource file
hInstance	

4.1.2.3 BITMAP::BITMAP (HDC *HDC,* INT *IWIDTH,* INT *IHEIGHT,* COLORREF *CRCOLOR* = RGB(0, 0))

Constructor that

PARAMETERS

hDC	The handle to the screen context device
iWidth	

PARAMETERS

iHeight	

crColor	

4.1.3MEMBER FUNCTION DOCUMENTATION

4.1.3.1 BOOL BITMAP::CREATE (HDC *HDC,* LPTSTR *SZFILENAME*)

Used to update the object with bitmap info if empty constructor was used.

PARAMETERS

hDC	The handle to the screen context device
szFileName	The file of the bitmap the Bitmap object will be created from

4.1.3.2 BOOL BITMAP::CREATE (HDC *HDC,* UINT *UIRESID,* HINSTANCE *HINSTANCE*)

Used to update the object with bitmap info if empty constructor was used.

PARAMETERS

hDC	The handle to the screen context device
uiResID	The ID of the bitmap from the resource file
hInstance	

4.1.3.3 BOOL BITMAP::CREATE (HDC *HDC,* INT *IWIDTH,* INT *IHEIGHT,* COLORREF *CRCOLOR*)

Used to update the object with bitmap info if empty constructor was used.

PARAMETERS

hDC	The handle to the screen context device
iWidth	
iHeight	
crColor	

4.1.4.4 VOID BITMAP::DRAW (HDC *HDC,* INT *X,* INT *Y,* BOOL *BTRANS* = FALSE, COLORREF *CRTRANSCOLOR* = RGB(255, 0, 255) **)**

Draws the bitmap to the screen

PARAMETERS

hDC	The handle to the screen context the bitmap will be drawn on
X	The X coordinate of the screen where to start drawing the bitmap
Y	The Y coordinate of the screen where to start drawing the bitmap
bTrans	If true, the bitmap has transparency. Default is no transparency
crTransColor	The color that will not be drawn to the screen to give transparency

4.1.4.5 VOID BITMAP::DRAWPART (HDC *HDC*, INT *X*, INT *Y*, INT *XPART*, INT *YPART*, INT *WPART*, INT *HPART*, BOOL *BTRANS*, COLORREF *CRTRANSCOLOR*)

Only draws part of a bitmap to the screen. Can be used for tilesheets or animation frames.

PARAMETERS

hDC	The handle to the screen context the bitmap will be drawn on
X	The X coordinate of the screen where to start drawing the bitmap
Y	The Y coordinate of the screen where to start drawing the bitmap
xPart	The X pixel of the bitmap to start drawing
yPart	The Y pixel of the bitmap to start drawing
wPart	The width in pixels used to draw from xPart
hPart	The height in pixels used to draw from yPart
bTrans	If true, the bitmap has transparency. Default is no transparency
crTransColor	The color that will not be drawn to the screen to give transparency

The documentation for this class was generated from the following files:

- seniorcapstone/CapstoneProject/CapstoneProject/Bitmap.h
- seniorcapstone/CapstoneProject/CapstoneProject/Bitmap.cpp

4.2ENEMYBOSS CLASS REFERENCE

Extends the Sprite class to give enemies extra variables and methods special to them.

#include <EnemyBoss.h>

Inherits Sprite.

PUBLIC MEMBER FUNCTIONS

- EnemyBoss (Bitmap ∗, RECT &, BOUNDSACTION)
- ∼EnemyBoss ()

  Empty.
- void SetHealth (int)
- int GetHealth () const

  Gets the how much health is left of the enemy.
- void MoveRandomly (int, int)
- void MoveChase (Sprite ∗)
- void MoveRunAway (Sprite ∗)

ADDITIONAL INHERITED MEMBERS

4.2.1DETAILED DESCRIPTION

Extends the Sprite class to give enemies extra variables and methods special to them.

This class extends the Sprite class to give enemies health and movements.

4.2.2CONSTRUCTOR & DESTRUCTOR DOCUMENTATION

4.2.2.1 ENEMYBOSS::ENEMYBOSS (BITMAP ∗ *PBITMAP,* RECT & *RCBOUNDS,* BOUNDSACTION *BABOUNDSACTION* = BA_STOP)

Creates an enemy sprite using the Bitmap/Sprite constructor

PARAMETERS

pBitmap	A pointer to the bitmap image
rcBounds	A rectangle to contain the movements of the sprite within it
baBoundsAction	The action that will be taken when the sprite hits the outter edge of the rcBounds rectangle

See also

Sprite

4.2.3MEMBER FUNCTION DOCUMENTATION

4.2.3.1 VOID ENEMYBOSS::MOVECHASE (SPRITE ∗ *PSPRITETOCHASE*)

Will have the sprite chase the Hero sprite\

PARAMETERS

pSpriteToChase	The pointer of the sprite you want this sprite to chase

4.2.3.2 VOID ENEMYBOSS::MOVERANDOMLY (INT *X,* INT *Y*)

Will have the sprite move randomly on the screen

PARAMETERS

x	Width of the area you want the sprite to move in
y	Height of the area you want the sprite to move in

4.2.3.3 VOID ENEMYBOSS::MOVERUNAWAY (SPRITE ∗ *PAVOIDSPRITE*)

Will have the sprite avoid another sprite

PARAMETERS

pAvoidSprite	The pointer of the Sprite to avoid

4.2.4.4 VOID ENEMYBOSS::SETHEALTH (INT *IHEALTH*)

Sets the health of the enemy so it can take multiple collisions before dying

PARAMETERS

iHealth	The health in which you want the enemy to have

The documentation for this class was generated from the following files:

- seniorcapstone/CapstoneProject/CapstoneProject/EnemyBoss.h
- seniorcapstone/CapstoneProject/CapstoneProject/EnemyBoss.cpp

4.3GAMEENGINE CLASS REFERENCE

The GameEngine class is used to encapsulate Window types, functions, and WinMain.

#include <GameEngine.h>

PUBLIC MEMBER FUNCTIONS

- GameEngine (HINSTANCE, LPTSTR, LPTSTR, WORD, WORD, int, int)
- virtual ∼GameEngine ()

  Changes the screen resolution back to what it was originally.
- bool Initialize (int)
- LRESULT HandleEvent (HWND, UINT, WPARAM, LPARAM)
- void AddLoadingSprite (Sprite ∗)
- void AddConstructSprite (Sprite ∗)
- void AddDungeonSprite (Sprite ∗) • void DrawLoadingSprites (HDC)
- void DrawConstructSprites (HDC)
- void DrawDungeonSprites (HDC)
- void UpdateLoadingSprites ()

  Expands the Sprite vector if necessary, updates the sprites positions, checks to see if a sprite was added or dying, and finally checks for collisions.
- void UpdateConstructSprites ()

  Expands the Sprite vector if necessary, updates the sprites positions, checks to see if a sprite was added or dying, and finally checks for collisions.
- void UpdateDungeonSprites ()

  Expands the Sprite vector if necessary, updates the sprites positions, checks to see if a sprite was added or dying, and finally checks for collisions.
- void CleanupSprites ()

  Transverses the Sprite vector to delete Sprite objects from memory.
- Sprite ∗ IsPointInSprite (int, int)
- HINSTANCE GetInstance () const

Returns the handle of the instance.

- HWND GetWindow () const

  Returns the handle of the window.

- void SetWindow (HWND)
- LPTSTR GetTitle ()

  returns the string of the game title

- WORD GetIcon () const

  returns the icon that will be shown in the task bar

- WORD GetSmallIcon () const

  returns the little icon that will be shown in the title bar

- int GetWidth () const *returns the width of the of the window*
- int GetHeight () const *returns the height of the window*
- int GetFrameDelay () const *returns the an interger value used to set a frame delay to slow the game down*
- void SetFrameRate (int)
- bool GetSleep () const

  Returns true if the game is paused.

- void SetSleep (bool)
- void SetScreenResolution ()

  Sets the screen resolution to 800x600.

STATIC PUBLIC MEMBER FUNCTIONS

- static GameEngine ∗ GetEngine ()

  Returns the pointer of the GameEngine.

Protected Member Functions

- bool CheckSpriteCollision (Sprite ∗)

PROTECTED ATTRIBUTES

- HINSTANCE m_hInstance

  Stores the handle to the instance of the GameEngine object.

- HWND m_hWindow

  Handle to the window.

- TCHAR m_szWindowClass [32]

  Used to store the name of the WindowClass that will register with Task Manager.

- TCHAR m_szTitle [32]

  Used to store the name of the title that will be displayed in the title bar of the window.

- WORD m_wIcon

  Used to hold the icon that will display on the task bar.

- WORD m_wSmallIcon

  Used to hold the icon that will display on the title bar of the window.

- int m_iWidth

  The width in pixels of the game size. NOT the window size.

- int m_iHeight

  The Height in pixels of the game size. NOT the window size.

- int m_iFrameDelay

  Used to slow down the GameEngine so humans can see what is happening on screen.

- bool m_bSleep

  Used to hold the status of the GameEngine. To pause or unpause.

- vector< Sprite ∗> m_vLoadingSprites

  A vector to hold all the sprites. Easier to cycle through a vector to determine Sprites status.

- vector< Sprite ∗> m_vConstructSprites

  A vector to hold all the sprites. Easier to cycle through a vector to determine Sprites status.

- vector< Sprite ∗> m_vDungeonSprites

  A vector to hold all the sprites. Easier to cycle through a vector to determine Sprites status.

- DEVMODE m_devmode

  Data structure containing info about the display device. Used to change screen resolution.

- long m_lResult

  Holds the results of the DEVMODE changes.

STATIC PROTECTED ATTRIBUTES

- static GameEngine ∗ m_pGameEngine = NULL

  The pointer to the GameEngine object.

4.3.1DETAILED DESCRIPTION

The GameEngine class is used to encapsulate Window types, functions, and WinMain.

Creates a static pointer to itself. Sets up the window class, the window title, icons, and dimensions of the screen. Uses a loop to cycle through sprite updates and draw them. Sets the timing of the game and keeps status of the game. Handles keyboard input.

4.3.2CONSTRUCTOR & DESTRUCTOR DOCUMENTATION

4.3.2.1 GAMEENGINE::GAMEENGINE (HINSTANCE *HINSTANCE,* LPTSTR *SZWINDOWCLASS,* LPTSTR *SZTITLE,* WORD *WICON,* WORD *WSMALLICON,* INT *IWIDTH* = 800, INT *IHEIGHT* = 640)

Creates the game window and sets the window parameters. Changes the screen resolution to 800x600.

PARAMETERS

hInstance	A handle to an instance. This is the base address of the module in memory.
szWindowClass	The name of the Window class that is registered to Task Manager
wIcon	The name of the icon to use from resources file. Big icon that will show in task bar.
wSmallIcon	The name of the icon to use from the resources file. Little icon that will show in the title bar of the window.
iWidth	The width of the game in pixels. NOT the window size.
iHeight	The height of the game in pixels. NOT the window size.

4.3.3MEMBER FUNCTION DOCUMENTATION

4.3.3.1 VOID GAMEENGINE::ADDCONSTRUCTSPRITE (SPRITE ∗ *PSPRITE*)

Pushes a Sprite object into the Sprite vector

PARAMETERS

A	pointer to the Sprite that will be added to the vector

See also m_vSprites

4.3.3.2 VOID GAMEENGINE::ADDDUNGEONSPRITE (SPRITE ∗ *PSPRITE*)

Pushes a Sprite object into the Sprite vector

PARAMETERS

A	pointer to the Sprite that will be added to the vector

See also m_vSprites

4.3.3.3 VOID GAMEENGINE::ADDLOADINGSPRITE (SPRITE ∗ *PSPRITE*)

Pushes a Sprite object into the Sprite vector

PARAMETERS

A	pointer to the Sprite that will be added to the vector

See also m_vSprites

4.3.4.4 BOOL GAMEENGINE::CHECKSPRITECOLLISION (SPRITE ∗ *PTESTSPRITE*) [PROTECTED]

Checks the Sprite object that is passed to it to determine if it has collided with another sprite or boundary

PARAMETERS

pTestSprite	A pointer of the Sprite to be tested

4.3.4.5VOID GAMEENGINE::DRAWCONSTRUCTSPRITES (HDC *HDC*)

Transverses the Sprite vector calling each Sprite's Draw function in the Construct level **Parameters**

hDC	Handle to the device context.

4.3.3.6 VOID GAMEENGINE::DRAWDUNGEONSPRITES (HDC *HDC*)

Transverses the Sprite vector calling each Sprite's Draw function in the dungeon level.

PARAMETERS

hDC	Handle to the device context.

4.3.3.7 VOID GAMEENGINE::DRAWLOADINGSPRITES (HDC *HDC*)

Transverses the Sprite vector calling each Sprite's Draw function on the loading screen

PARAMETERS

hDC	Handle to the device context.

4.3.3.8LRESULT GAMEENGINE::HANDLEEVENT (HWND *HWINDOW,* UINT *MSG,* WPARAM *WPARAM,* LPARAM *LPARAM*)

Routes Windows messages to GameEngine member functions. If message isn't used by GameEngine, it is passed back to Windows for handling.

PARAMETERS

hWindow	The handle to the window
msg	The message that is being passed
wParam	A parameter for which key is being pressed in keyboard messages
lParam	Not used. Will be passed back to Windows

4.3.3.9BOOL GAMEENGINE::INITIALIZE (INT *ICMDSHOW*)

Used to setup the window and put the focus on it. Tests the results and if everything passes true that was loaded the game continues. If something fails, the game will terminate with an error.

PARAMETERS

iCmdShow	Sets how the window should be shown

SEE ALSO

Microsoft's MSDN site for iCmdShow values: https://goo.gl/fIHb60

4.3.3.10 SPRITE ∗ GAMEENGINE::ISPOINTINSPRITE (INT *X*, INT *Y*)

Transverses the Sprite vector to see if the given point is within any of the Sprites

PARAMETERS

x	The X coordinate that will be checked
y	The Y coordinate that will be checked

4.3.3.11 VOID GAMEENGINE::SETFRAMERATE (INT *IFRAMERATE*)

Used to set how fast the game should update on the screen

PARAMETERS

iFrameRate	This is used to divide 1000ms

4.3.3.12 VOID GAMEENGINE::SETSLEEP (BOOL *BSLEEP*)

If the game is paused, set sleep to true.

PARAMETERS

bSleep	True if the game is paused

4.3.3.13 VOID GAMEENGINE::SETWINDOW (HWND *HWINDOW*)

Sets the handle of the window

PARAMETERS

hWnd	A handle to the window

The documentation for this class was generated from the following files:

- seniorcapstone/CapstoneProject/CapstoneProject/GameEngine.h
- seniorcapstone/CapstoneProject/CapstoneProject/GameEngine.cpp

4.4HERO CLASS REFERENCE

Extends the Sprite class to give the hero extra variables and methods special to it.

#include <Hero.h>

Inherits Sprite.

PUBLIC MEMBER FUNCTIONS

- Hero (Bitmap ∗, RECT &, BOUNDSACTION)

  All the bitmaps that are used by the Hero.
- ~Hero ()

  Empty.
- void MoveLeft ()

  Changes the Sprite's velocity to move left on the screen and associated bitmap.
- void MoveRight ()

  Changes the Sprite's velocity to move right on the screen and associated bitmap.
- void MoveUp ()

  Changes the Sprite's velocity to move up on the screen and associated bitmap.
- void MoveDown ()

  Changes the Sprite's velocity to move down on the screen and associated bitmap.
- void SetWeapon (WEAPONTYPE)
- void SetHealth (int)
- int GetHealth () const
- Sprite ∗ UseWeapon ()
- void IncreaseHealthMeter ()

  Used to increase the health meter and refill the Sprite to full health.
- WEAPONTYPE GetWeapon () const
- FACINGDIRECTION GetDirection () const
- void SetDirection (FACINGDIRECTION)

Additional Inherited Members

4.4.1DETAILED DESCRIPTION

Extends the Sprite class to give the hero extra variables and methods special to it.

This class extends the Sprite class to give the hero health, weapon choice and movements.

4.4.2CONSTRUCTOR & DESTRUCTOR DOCUMENTATION

4.4.2.1HERO::HERO (BITMAP ∗ *PBITMAP,* RECT & *RCBOUNDS,* BOUNDSACTION *BABOUNDSACTION* = BA_STOP)

All the bitmaps that are used by the Hero.

Creates an enemy sprite using the Bitmap/Sprite constructor

PARAMETERS

pBitmap	A pointer to the bitmap image
rcBounds	A rectangle to contain the movements of the sprite within it
baBoundsAction	The action that will be taken when the sprite hits the outter edge of the rcBounds rectangle

See also

Sprite

4.4.3MEMBER FUNCTION DOCUMENTATION

4.4.3.1 FACINGDIRECTION HERO::GETDIRECTION ( ) CONST

Returns the direction in which the hero is facing

Returns

m_fdDirection as a WORD

SEE ALSO

FACINGDIRECTION

4.4.3.2INT HERO::GETHEALTH ( ) CONST

Returns how much health the sprite has left before dying

Returns m_iHealth as an integer

4.4.3.3 WEAPONTYPE HERO::GETWEAPON ( ) CONST

Returns which weapon the hero is currently holding

Returns

m_wtWeapon as a WORD

SEE ALSO

WEAPONTYPE

4.4.3.4 VOID HERO::SETDIRECTION (FACINGDIRECTION *FDDIRECTION*)

Sets the direction the hero is facing

PARAMETERS

fdDirection	The direction to set the member variable to

4.4.3.5 VOID HERO::SETHEALTH (INT *IHEALTH*)

Sets the health of the hero

PARAMETERS

iHealth	The health you want the hero to have but can't be more than the m_iHealthMeter

4.4.3.6 VOID HERO::SETWEAPON (WEAPONTYPE *WTWEAPON*)

Sets the weapon type that the hero will use in game play

PARAMETERS

wtWeapon	The weapon type to be stored

SEE ALSO

WEAPONTYPE

4.4.3.7SPRITE ∗ HERO::USEWEAPON ( )

Will change the sprites animation to show weapon usage and will also show it has the hitter in collision detection. This will make sure that the hero sprite doesn't lose health or die while weapon is in use.

The documentation for this class was generated from the following files:

- seniorcapstone/CapstoneProject/CapstoneProject/Hero.h
- seniorcapstone/CapstoneProject/CapstoneProject/Hero.cpp

4.5SPRITE CLASS REFERENCE

Takes a bitmap and turns it into a sprite with position, velocity, z-order, bounding rect, collision, and visiblity.

#include <Sprite.h>

Inherited by EnemyBoss, and Hero.

PUBLIC MEMBER FUNCTIONS

- Sprite (Bitmap ∗)
- Sprite (Bitmap ∗, RECT &, BOUNDSACTION baBoundsAction=BA_STOP)
- Sprite (Bitmap ∗, POINT, POINT, int, RECT &, BOUNDSACTION baBoundsAction=BA_STOP)
- virtual ~Sprite ()

  Empty destructor, nothing special here.
- virtual SPRITEACTION Update ()
- void Draw (HDC)
- bool IsPointInside (int, int)
- bool TestCollision (Sprite ∗)
- void Kill ()

  Sets the sprite's member variable to dying.
- Bitmap ∗ GetBitmap () const
- void SetNumFrames (int, bool bOneCycle=false)
- void SetFrameDelay (int)

  How long the sprite will stay on one frame before switching to the next frame.
- void SetBitmap (Bitmap ∗)
- RECT & GetPosition ()

  Returns the position of the sprite by using a rectangle around the frame of the bitmap.
- void SetPosition (int, int)
- void SetPosition (POINT)
- void SetPosition (RECT &)
- void OffsetPosition (int, int)
- RECT & GetCollision ()

  Returns the smaller collision rectangle that is around the sprite.
- POINT GetVelocity ()

  Returns the velocity of the sprite in the terms of a POINT.
- void SetVelocity (int, int)
- void SetVelocity (POINT)
- bool GetZorder () const

  Returns if the sprite has a Z-order that is used in layering.
- void SetZorder (int)
- void SetBounds (RECT &)
- void SetBoundsAction (BOUNDSACTION)
- bool IsHidden () const

  Returns true if the spirte is hidden from the screen.
- void SetHidden (bool)

- int GetWidth () const

  Returns the width of the bitmap/sprite.

- int GetHeight () const

  Returns the height of the bitmap/sprite.

PUBLIC ATTRIBUTES

- Sprite ∗ AddSprite

  A pointer to this sprite object.

PROTECTED MEMBER FUNCTIONS

- void UpdateFrame ()

  Used to change from the current frame of animation to the next frame.

- virtual void CalcCollisionRect ()

  Calculates the collision rectangle to be one-sixth smaller than the position rectangle using the X and Y dimensions of the sprite.

PROTECTED ATTRIBUTES

- Bitmap ∗ m_pBitmap

  Pointer to the bitmap used to create the sprite.

- int m_iNumFrames

  How many frames are in the bitmap to create animation for the sprite.

- int m_iCurFrame

  Used to specify the current frame of the bitmap when animation is used.

- int m_iFrameDelay

  How long you the game needs to wait before changing to the next frame of animation.

- int m_iFrameTrigger

  A counter that is compated to the delay that will trigger the next frame.

- int m_iZorder

  Used if you have to layer bitmaps or spirtes on top of each other.

- POINT m_ptVelocity

  Used in caluclations to move the sprite around the screen.

- RECT m_rcPosition

  The rectangle around the bitmap that is the position of the sprite.

- RECT m_rcCollision

  A slightly smaller rectangle used for in collision detection calculations.

- RECT m_rcBounds

  The rectangle that is usually larger than the sprite that will contain the movements of the sprite within this rectangle.

- BOUNDSACTION m_baBoundsAction

Holds the current BOUNDSACTION assigned to the sprite.

- bool m_bHidden

  Used to show or hide the sprite on the screen.

- bool m_bDying

  Used to indicate if the sprite is dying.

- bool m_bOneCycle

  Used to state if the sprites animation will run once or loop.

4.5.1DETAILED DESCRIPTION

Takes a bitmap and turns it into a sprite with position, velocity, z-order, bounding rect, collision, and visiblity.

This class depends on Bitmap to function. Does not extend Bitmap. There are three different types of constructors used to create a sprite. This class controls how the sprite moves, how the sprite will interact with its boundries. It will either stop, wrap, bounce, or die at the boundry. There are also three SPRITEACTIONs that set what the sprite is doing.

4.5.2CONSTRUCTOR & DESTRUCTOR DOCUMENTATION

4.5.2.1 SPRITE::SPRITE (BITMAP ∗ *PBITMAP*)

Creates a generic sprite using the Bitmap constructor.

PARAMETERS

pBitmap	A pointer to the bitmap object

SEE ALSO

Bitmap

4.5.2.2 SPRITE::SPRITE (BITMAP ∗ *PBITMAP,* RECT & *RCBOUNDS,* BOUNDSACTION *BABOUNDSACTION* = BA_STOP)

Creates a sprite using the Bitmap constructor but adds additional attributes.

PARAMETERS

pBitmap	A pointer to the bitmap object
rcBounds	A rectangle to contain the movements of the sprite within it
baBoundsAction	The action that will be taken when the sprite hits the outter edge of the rcBounds rectangle. Default is for the sprite to stop.

SEE ALSO

Bitmap

4.5.2.3 SPRITE::SPRITE (BITMAP ∗ *PBITMAP,* POINT *PTPOSITION,* POINT *PTVELOCITY,* INT *IZORDER,* RECT & *RCBOUNDS,* BOUNDSACTION *BABOUNDSACTION* = BA_STOP)

Creates a sprite using the Bitmap constructor but adds additional attributes.

PARAMETERS

pBitmap	A pointer to the bitmap object
ptPosition	Sets a point of where the sprite will be positioned on the screen
ptVelocity	Sets the velocity of the sprite to move
iZOrder	Sets what layer the sprite needs to be if there will be over lapping of sprites
rcBounds	A rectangle to contain the movements of the sprite within it
baBoundsAction	The action that will be taken when the sprite hits the outter edge of the rcBounds rectangle. Default is for the sprite to stop.

See also

Bitmap

4.5.3MEMBER FUNCTION DOCUMENTATION

4.5.3.1 VOID SPRITE::DRAW (HDC *HDC*)

Checks the hidden status and current frame, then draw appropriate image to the screen.

PARAMETERS

hDC	The handle to the display context of where to draw the image.

4.5.3.2 BITMAP ∗ SPRITE::GETBITMAP ( ) CONST

Pointer to the Bitmap of the sprite

SEE ALSO

Bitmap

4.5.3.3 BOOL SPRITE::ISPOINTINSIDE (INT *X,* INT *Y*)

Determines if a given point, in the terms of X and Y, are inside a specified area.

PARAMETERS

x	coordinate X

y	coordinate Y

4.5.4.4 VOID SPRITE::OFFSETPOSITION (INT *X,* INT *Y*)

Used to offset the position of a sprite by using X and Y pixels

PARAMETERS

X	Number of X pixels used to offset sprite horizontally
Y	Number of Y pixels used to offset sprite vertically

4.5.4.5 VOID SPRITE::SETBITMAP (BITMAP ∗ *PBITMAP*)

Used to change the Bitmap image that the sprite uses.

PARAMETERS

pBitmap	The pointer to the bitmap

SEE ALSO

Bitmap

4.5.3.6VOID SPRITE::SETBOUNDS (RECT & *RCBOUNDS*)

Sets a rectangle that is used to allow the movement of the sprite within the rectangle **Parameters**

rcBounds	The rectangle that will be set around the sprite

4.5.3.7 VOID SPRITE::SETBOUNDSACTION (BOUNDSACTION *BA*)

Sets the bounds action of the sprite. Sprite will either stop, bounce back, or wrap to the other side of the screen.

PARAMETERS

baBoundsAction	The action you want to set to the sprite

SEE ALSO

BOUNDSACTION

4.5.3.8VOID SPRITE::SETHIDDEN (BOOL *BHIDDEN*)

Used to show or hide the sprite.

PARAMETERS

bHidden	If true, the sprite will not display on the screen

4.5.3.9 VOID SPRITE::SETNUMFRAMES (INT *INUMFRAMES,* BOOL *BONECYCLE* = FALSE)

Used to set how many frames to use for sprite animation.

PARAMETERS

iNumFrames	how many frames there are
bOneCycle	Used to determine if the animation should loop. Default is to loop.

4.5.3.10 VOID SPRITE::SETPOSITION (INT *X*, INT *Y*)

X	The X coordinate
Y	The Y coordinate

Sets the position of the sprite on the screen using X and Y coordinates.

PARAMETERS

4.5.3.11 VOID SPRITE::SETPOSITION (POINT *PTPOSITION*)

Sets the position of the sprite on the screen using the type POINT.

PARAMETERS

ptPosition	The position of the sprite using a POINT type.

4.5.3.12 VOID SPRITE::SETPOSITION (RECT & *RCPOSITION*)

Sets the position of the sprite on the screen using a rectangle

PARAMETERS

rcPosition	The positions of the rectangle

4.5.3.13 VOID SPRITE::SETVELOCITY (INT *X*, INT *Y*)

Sets the velocity of the spirte by using X and Y pixels

PARAMETERS

X	The number of pixels you need the sprite to move horizontally every update
Y	The number of pixels you need the sprite to move vertically every update

4.5.3.14 VOID SPRITE::SETVELOCITY (POINT *PTVELOCITY*)

Sets the velocity of the sprite by using a POINT type

PARAMETERS

ptVelocity	

4.5.3.15 VOID SPRITE::SETZORDER (INT *IZORDER*)

Sets the Z-order of the sprite in layered sprites

PARAMETERS

iZOrder	A number used to set the which layer you want the sprite on

4.5.3.16 BOOL SPRITE::TESTCOLLISION (SPRITE ∗ *PTESTSPRITE*)

Returns true if the Sprite runs into its boundry rectangle.

PARAMETERS

pTestSprite	pointer to the sprite that needs to be tested

4.5.3.17SPRITEACTION SPRITE::UPDATE ( ) [VIRTUAL]

Determines the current status of the sprite and takes action on what needs to be done next. If the sprite is dying, it will KILL the sprite. Will check the BOUNDSACTION and update velocity, bounce it, or wrap it.

SEE ALSO

SPRITEACTION

BOUNDSACTION

The documentation for this class was generated from the following files:

- seniorcapstone/CapstoneProject/CapstoneProject/Sprite.h
- seniorcapstone/CapstoneProject/CapstoneProject/Sprite.cpp