GNOME PRINCE VS THE BLADE SKELETON (SOURCE CODE)

SECTION 1 – CAVE MANAGER CLASS

```
package
{
	import flash.display.MovieClip;
	import flash.events.Event;
	import flash.events.MouseEvent;

	// timer event preparation for freeze

	import flash.events.TimerEvent;
	import flash.utils.Timer;
	import flash.media.SoundChannel;
	import flash.media.SoundTransform;


	public class CaveManager extends MovieClip
	{

		// arrays to store the arrow and projectile ammo count etc

		private var _arrows:uint;
		private var _gpHasBowInv:Boolean;
```

```
private var _gpHasSwordInv:Boolean;
private var _gpHasSpellInv:Boolean;

private var _gnInvArray:Array;
private var _gnInvNumber:Number;
private var _gnInvStatus:Boolean;
private var _inventorySlotPlaceholder:Array;
private var _skelVx:Number;
private var _skelVy:Number;
private var _skelTrav:int;
private var _binaryRand:int;
private var _freezeTimer:Timer;
private var _freezeStatus:Boolean;
private var _skelHp:int;

// private variable for sound status
private var _soundActive:Boolean;

// private variable for splash screen
private var _splashActive:Boolean;
private var _gameBegin:Boolean;

private var _menuActive:Boolean;

private var _tutorialActive:Boolean;
```

```
private var _gameOverStatus:Boolean;
private var _gameVictoryStatus:Boolean;
// death animation temp holder
private var _deathLocationX:int;
private var _deathLocationY:int;

// CHECK FOR STANDARD enemy hits
private var _enemyHit:Boolean;

// create an enemy hit timer
private var _enemyHitTimer:Timer;
// setup weapon delay
private var _quivTimer:Timer;
private var _frostTimerDelay:Timer;
private var _swordDelayTimer:Timer;

// setup game active mode
private var _gameActive:Boolean;

// This code was made by Tyler Stansfield Jaggers - Overidon.com July, 2013
// random critical damage
private var _critRand:uint;

// gnome prince stat variables
private var _gpHp:int;
```

```
private var _gpWeaponDamage:int;
private var _gpCriticalStrike:int;

// create variable to randomize skel movement
// after an inventory press

private var _skelRandInv:int;

// private temporary variables for inventory situation
private var _tempVx:int;
private var _tempVy:int;
private var _tempX:int;
private var _tempY:int;

private var _projectiles:Array;

var blinkInvBox0:BlinkInvBox;
var blinkInvBox1:BlinkInvBox;
var blinkInvBox2:BlinkInvBox;
var quivEquip:Bow;
var swordEquip:Sword;
var spellEquip:SpellPickup;
var bladeSkel1:BladeSkel;
var pubGnVx:int;
var deathAnimation:DeathAnimation;
```

```
// item pickups
var spellPickup:SpellPickup;
var swordPickup:Sword;
var bowPickup:Bow;

// sounds
var frostHit:FrostHit;
var enemyHit:EnemyHit;
var gameOverSound:GameOverSound;
var gnTutorial:GnTutorial = new GnTutorial();
var gnCombat:GnCombat = new GnCombat();
var gnomeVictory:GnomeVictory = new GnomeVictory();
var myChannel:SoundChannel = new SoundChannel();
var myTransform:SoundTransform = new SoundTransform();
var combatTransform:SoundTransform = new SoundTransform();
var musicChannel:SoundChannel = new SoundChannel();
var victoryChannel:SoundChannel = new SoundChannel();

// menu and splash modules
var tutorial:Tutorial;
var mainMenu:MainMenu;
var splashScreen:SplashScreen;

var blackBoxCover0:BlackBoxCover;
```

```
var blackBoxCover1:BlackBoxCover;
var blackBoxCover2:BlackBoxCover;

public function CaveManager()
{
	init();
}

function init():void
{
	dispatchEvent (new Event("soundEnabled", true));

	// add the pickup for spell
	spellPickup = new SpellPickup();
	this.addChild(spellPickup);
	spellPickup.x = 375;
	spellPickup.y = 275;
	//sword
	swordPickup = new Sword();
	this.addChild(swordPickup);
	swordPickup.x = 475;
	swordPickup.y = 125;
	// bow
	bowPickup = new Bow();
	this.addChild(bowPickup);
```

```
bowPickup.x = 475;
bowPickup.y = 225;

// make the gnome visible
gnomePrince.visible = true;
gnomePrince.x = 55;
gnomePrince.y = 100;
// set sound to "ON" or "true"
_soundActive = true;

// prep the pre-game menus and splash etc
_splashActive = true;
splashScreen = new SplashScreen();
this.addChild(splashScreen);
splashScreen.x = 275;
splashScreen.y = 225;
this.splashScreen.visible = true;



_gameBegin = false;
_gameActive = false;
_menuActive = false;
mainMenu = new MainMenu();
this.addChild(mainMenu);
mainMenu.x = 275;
```

```
mainMenu.y = 225;
this.mainMenu.visible = false;

_tutorialActive = false;
tutorial = new Tutorial();
this.addChild(tutorial);
tutorial.x = 275;
tutorial.y = 225;
this.tutorial.visible = false;
this.tutorial.stop();

_gameOverStatus = false;
this.gameOver.visible = false;
_gameVictoryStatus = false;
this.congrats.visible = false;
this.congrats.stop();

// initialize the inventory mode visibility to false
this.inventoryMaster.visible = false;

this.inventoryMaster.frostAppearance.visible = false;
this.inventoryMaster.swordAppearance.visible = false;
this.inventoryMaster.quivAppearance.visible = false;

// initialize gp stats
```

```
_gpHp = 100;
_gpWeaponDamage = 0;
_gpCriticalStrike = 0;

// display these stats in the inventory screent
this.inventoryMaster.hpBox.text = _gpHp + "/100";
this.inventoryMaster.weaponDamageBox.text = String(_gpWeaponDamage);
this.inventoryMaster.criticalStrikeBox.text = String(_gpCriticalStrike);

// initialize arrow ammo total
_arrows = 0;
_gpHasBowInv = false;
_gpHasSwordInv = false;
_gpHasSpellInv = false;

// add Skeleton Hit points
_skelHp = 400;

// initialize the freezeDisplay
_freezeTimer = new Timer(1000);
// add the freezeStatus modifier boolean
_freezeStatus = false;

// initialize the enemy hit timer
_enemyHitTimer = new Timer(100);
```

```
// initialize the enemy hit status
_enemyHit = false;

// setup the _quivTimer
_quivTimer = new Timer(100);
_quivTimer.start();

// setup the frost delay
_frostTimerDelay = new Timer(100);
_frostTimerDelay.start();

// setup the sword delay
_swordDelayTimer = new Timer(100);
_swordDelayTimer.start();

_gnInvArray = new Array();
_gnInvArray = ["QUIV", "SWORD", "SPELL"];
_gnInvNumber = 0;
_gnInvStatus = false;

// initialize the _projectiles array
_projectiles = new Array();
_projectiles = [];

blinkInvBox0 = new BlinkInvBox();
```

```
blinkInvBox1 = new BlinkInvBox();
blinkInvBox2 = new BlinkInvBox();
this.inventoryMaster.inventorySlot0.addChild(blinkInvBox0);
this.inventoryMaster.inventorySlot1.addChild(blinkInvBox1);
this.inventoryMaster.inventorySlot2.addChild(blinkInvBox2);

// I am proud if this little fix
blackBoxCover0 = new BlackBoxCover();
blackBoxCover1 = new BlackBoxCover();
blackBoxCover2 = new BlackBoxCover();
this.inventoryMaster.inventorySlot0.addChild(blackBoxCover0);
this.inventoryMaster.inventorySlot1.addChild(blackBoxCover1);
this.inventoryMaster.inventorySlot2.addChild(blackBoxCover2);

blinkInvBox0.visible = true
blinkInvBox1.visible = false;
blinkInvBox2.visible = false;

spellEquip = new SpellPickup();
swordEquip = new Sword();
quivEquip = new Bow();
this.inventoryMaster.equipArea.addChild(spellEquip);
this.inventoryMaster.equipArea.addChild(swordEquip);
this.inventoryMaster.equipArea.addChild(quivEquip);
```

```
                spellEquip.visible = false;
                swordEquip.visible = false;
                quivEquip.visible = false;

                // add an event listner for EnterFrame
                addEventListener(Event.ENTER_FRAME, onEnterFrame);

                // add listener for teh projectile events
                stage.addEventListener("projectileCreated", onProjectileCreated);
                stage.addEventListener("gnInvPressed", onGnInvPressed);
                // this "gnInvQuiv" listener is used once in CaveManager and also in other
classes
                // this is an attempt to use one dispatch for different purposes in different
classes
                stage.addEventListener("gnInvQuiv", onGnInvQuiv);
                // this "gnInvSword" listener is similar to the quiver
                // use it to dispatch the pickup of the sword
                stage.addEventListener("gnInvSword", onGnInvSword);
                // this "gnInvSpell" listener is for the spell
                stage.addEventListener("gnInvSpell", onGnInvSpell);
                stage.addEventListener("gnInvPressedLeft", onGnInvPressedLeft);
                stage.addEventListener("gnInvPressedRight", onGnInvPressedRight);
                stage.addEventListener("gnInvSelected", onGnInvSelected);
                // add a listener for the quiver delay issue
                stage.addEventListener("quivDelayCheck", onQuivDelayCheck);
                // add a listener for the frost delay issue
```

```
                stage.addEventListener("frostDelayCheck", onFrostDelayCheck);
                // add a listener for the sword delay issue
                stage.addEventListener("swordDelayCheck", onSwordDelayCheck);
                // add an event listener for the freezeDisplay
                _freezeTimer.addEventListener(TimerEvent.TIMER, onUpdateTime);
                // add an event listener for the enemy hit timer
                _enemyHitTimer.addEventListener(TimerEvent.TIMER, onUpdateHitTime);
                // add delay timers
                _quivTimer.addEventListener(TimerEvent.TIMER, onUpdateQuivTimer);
                _frostTimerDelay.addEventListener(TimerEvent.TIMER,
onUpdateFrostDelayTimer);
                _swordDelayTimer.addEventListener(TimerEvent.TIMER,
onUpdateSwordDelayTimer);

                // add an event listener for gameOver in the document class
                stage.addEventListener("gameOver", onGameOver);
                // add an event listener for the game victory in the document class
                stage.addEventListener("gameVictory", onGameVictory);

                // add event listeners for the button press
                upButton.addEventListener(MouseEvent.MOUSE_DOWN, onUpButtonClick);
                upButton.addEventListener(MouseEvent.MOUSE_UP, onUpButtonRelease);
                downButton.addEventListener(MouseEvent.MOUSE_DOWN,
onDownButtonClick);
                downButton.addEventListener(MouseEvent.MOUSE_UP,
onDownButtonRelease);
                leftButton.addEventListener(MouseEvent.MOUSE_DOWN, onLeftButtonClick);
```

```
                leftButton.addEventListener(MouseEvent.MOUSE_UP, onLeftButtonRelease);

                rightButton.addEventListener(MouseEvent.MOUSE_DOWN,
onRightButtonClick);

                rightButton.addEventListener(MouseEvent.MOUSE_UP, onRightButtonRelease);

                spaceBarButton.addEventListener(MouseEvent.MOUSE_DOWN,
onSpaceBarButtonClick);

                spaceBarButton.addEventListener(MouseEvent.MOUSE_UP,
onSpaceBarButtonRelease);

                inventoryButton.addEventListener(MouseEvent.MOUSE_DOWN,
onInventoryButtonClick);

                inventoryButton.addEventListener(MouseEvent.MOUSE_UP,
onInventoryButtonRelease);


                // add an event listener for the splash screen click

                splashScreen.addEventListener(MouseEvent.CLICK, onSplashScreenClick);

                // add an event listener for the game begin

                stage.addEventListener("gameBegin", onGameBegin);

                this.mainMenu.beginGameButton.addEventListener(MouseEvent.CLICK,
onBeginGameButtonClick);

                // add an event listener for the tutorial begin click

                this.mainMenu.tutorialButton.addEventListener(MouseEvent.CLICK,
onBeginTutorialClick);

                // add an event listener for the tutorial screen click

                this.tutorial.addEventListener(MouseEvent.CLICK, onTutorialExitClick);

                // add event listener for the game over screen

                this.gameOver.addEventListener(MouseEvent.CLICK, onGameOverClick);

                // add an event listener for the game victory screen

                this.congrats.addEventListener(MouseEvent.CLICK, onCongratsClick);
```

```
            soundButton.addEventListener(MouseEvent.CLICK, onSoundButtonClick);
}
private function onCongratsClick(event:MouseEvent):void
{
            init();
            victoryChannel.stop();
}
private function onGameOverClick(event:MouseEvent):void
{
            init();
}
private function onGameVictory(event:Event):void
{
            stage.removeEventListener("gameVictory", onGameVictory);
            _gameVictoryStatus = true;

            this.congrats.visible = true;
            this.congrats.play();



            if (bladeSkel1 != null)
            {
                        removeChild(bladeSkel1);
                        bladeSkel1 = null;
```

```
}
if (_soundActive == false)
{
	musicChannel.stop();
}
else if (_soundActive)
{
	musicChannel.stop();
	gnomeVictory = new GnomeVictory();

	victoryChannel = gnomeVictory.play(0, 1);
	myTransform.volume = 0.5;
	victoryChannel.soundTransform = myTransform;
}
// invis some stuff
if (spellPickup != null)
{
	spellPickup.visible = false;
}
if (bowPickup != null)
{
	bowPickup.visible = false;
}
if (swordPickup != null)
{
```

```
                swordPickup.visible = false;
        }
        // remove items on the stage

        // remove the old equppable inv itams :D
        spellEquip.visible = false;
        swordEquip.visible = false;
        quivEquip.visible = false;
        _gpHasBowInv = false;
        _gpHasSwordInv = false;
        _gpHasSpellInv = false;
        // initialize the inventory mode visibility to false
        this.inventoryMaster.visible = false;
}
private function onGameOver(event:Event):void
{
        this.gameOver.visible = true;

        stage.removeEventListener("gameOver", onGameOver);
        if (congrats.visible == true)
        {
                congrats.visible = false;
        }
        _gameOverStatus = true;
```

```
if (bladeSkel1 != null)
{
	removeChild(bladeSkel1);
	bladeSkel1 = null;
}
if (_soundActive == false)
{
	musicChannel.stop();
}
else if (_soundActive)
{
	musicChannel.stop();
	gameOverSound = new GameOverSound();
	gameOverSound.play();
}
// invis some stuff
if (spellPickup != null)
{
	spellPickup.visible = !spellPickup.visible;
}
if (bowPickup != null)
{
	bowPickup.visible = !bowPickup.visible;
}
if (swordPickup != null)
```

```
		{
			swordPickup.visible = !swordPickup.visible;
		}
		this.inventoryMaster.unarmedAppearance.visible = true;
		// remove the old blink boxes
		this.inventoryMaster.inventorySlot0.removeChild(blinkInvBox0);
		this.inventoryMaster.inventorySlot1.removeChild(blinkInvBox1);
		this.inventoryMaster.inventorySlot2.removeChild(blinkInvBox2);
		// remove the old equppable inv itams :D
		spellEquip.visible = false;
		swordEquip.visible = false;
		quivEquip.visible = false;
		_gpHasBowInv = false;
		_gpHasSwordInv = false;
		_gpHasSpellInv = false;
	}
	private function onSoundButtonClick(event:MouseEvent)
	{
		_soundActive = !_soundActive;
		if (_soundActive == false)
		{
			musicChannel.stop();
		}
		else if (_soundActive == true)
		{
```

```
                musicChannel = gnCombat.play(0, 999);
                combatTransform.volume = 0.3;
                musicChannel.soundTransform = combatTransform;
        }
        dispatchEvent(new Event("soundToggle",true));
}
function onSoundChannelSoundComplete(e:Event):void
{
        musicChannel = gnCombat.play(100);
}
// make the splash screen disappear
private function onSplashScreenClick(event:MouseEvent):void
{
        _splashActive = false;
        this.splashScreen.visible = false;
        removeChild(splashScreen);
        this.splashScreen = null;
        // make the main menu visible
        this.mainMenu.visible = true;
        _menuActive = true;
        // create and play tutorial song
        if (_menuActive == true)
        {
                myChannel = gnTutorial.play(100);
                myTransform.volume = 0.5;
```

```
                myChannel.soundTransform = myTransform;
            }
        }
        private function onBeginTutorialClick(event:MouseEvent):void
        {
            _menuActive = false;
            this.mainMenu.visible = false;
            this.tutorial.visible = true;
            this.tutorial.play();
        }
        private function onTutorialExitClick(event:MouseEvent):void
        {
            this.tutorial.visible = false;
            removeChild(tutorial);
            myChannel.stop();
            dispatchEvent(new Event("gameBegin",true));
        }
        private function onBeginGameButtonClick(event:MouseEvent):void
        {
            _menuActive = false;
            this.mainMenu.visible = false;
            if (_menuActive == false)
            {
                // Stop the current music
                myChannel.stop();
```

```
            }

            removeChild(mainMenu);
            this.mainMenu = null;
            dispatchEvent(new Event("gameBegin",true));

}
// begin game
private function onGameBegin(event:Event):void
{
            _gameActive = true;
            if (_gameActive == true)
            {
                        musicChannel = gnCombat.play(0, 999);
                        combatTransform.volume = 0.3;
                        musicChannel.soundTransform = combatTransform;
            }

            // stop the tutorial song
            _menuActive = false;
            // make bad guy
            bladeSkel1 = new BladeSkel();
            this.addChild(bladeSkel1);
            bladeSkel1.x = 140;
            bladeSkel1.y = 266;
```

```
            _skelVx = 7;
            _skelVy = 0;
            _skelTrav = 0;

            bladeSkel1.frozeSkel.visible = false;
            bladeSkel1.hurtSkel.visible = false;
}
// massive dispatching :D
private function onInventoryButtonClick(event:MouseEvent):void
{
            dispatchEvent(new Event("inventoryButtonClicked",true));
}
private function onInventoryButtonRelease(event:MouseEvent):void
{
            dispatchEvent(new Event("inventoryButtonReleased",true));
}
private function onSpaceBarButtonClick(event:MouseEvent):void
{
            dispatchEvent(new Event("spaceBarButtonClicked", true));
}
private function onSpaceBarButtonRelease(event:MouseEvent):void
{
            dispatchEvent(new Event("spaceBarButtonReleased", true));
}
private function onUpButtonClick(event:MouseEvent):void
```

```
{
        dispatchEvent(new Event("upButtonClicked", true));
}
private function onUpButtonRelease(event:MouseEvent):void
{
        dispatchEvent(new Event("upButtonReleased", true));
}
private function onDownButtonClick(event:MouseEvent):void
{
        dispatchEvent(new Event("downButtonClicked", true));
}
private function onDownButtonRelease(event:MouseEvent):void
{
        dispatchEvent(new Event("downButtonReleased", true));
}
private function onLeftButtonClick(event:MouseEvent):void
{
        dispatchEvent(new Event("leftButtonClicked" , true));
}
private function onLeftButtonRelease(event:MouseEvent):void
{
        dispatchEvent(new Event("leftButtonReleased", true));
}
private function onRightButtonClick(event:MouseEvent):void
{
```

```
        dispatchEvent(new Event("rightButtonClicked", true));
}
private function onRightButtonRelease(event:MouseEvent):void
{
        dispatchEvent(new Event("rightButtonReleased", true));
}
// sword delay checks
private function onSwordDelayCheck(event:Event):void
{
        if (_swordDelayTimer.currentCount == 5)
        {
                dispatchEvent(new Event("swordDrawnOk", true));
                _swordDelayTimer.reset();
                _swordDelayTimer.start();
        }
        else if (_swordDelayTimer.currentCount <=4)
        {
                dispatchEvent(new Event("swordDrawnNotReady", true));
        }
}

// quiver delay checks
private function onQuivDelayCheck(event:Event):void
{
        if (_quivTimer.currentCount == 6)
```

```
		{
			dispatchEvent(new Event("quivDrawnOk" , true));
			_quivTimer.reset();
			_quivTimer.start();
		}
		else if (_quivTimer.currentCount <= 5)
		{
			dispatchEvent(new Event("quivDrawnNotReady", true));
		}
	}
	private function onFrostDelayCheck(event:Event):void
	{
		if (_frostTimerDelay.currentCount == 9)
		{
			dispatchEvent(new Event("frostDrawnOk" , true) );
			_frostTimerDelay.reset();
			_frostTimerDelay.start();
		}
		else if (_frostTimerDelay.currentCount <= 8)
		{
			dispatchEvent(new Event("frostDrawnNotReady" , true));
		}
	}

	// begin the timer code
```

```
private function onUpdateSwordDelayTimer(event:Event):void
{
	if (_swordDelayTimer.currentCount == 5)
	{
		_swordDelayTimer.stop();
	}
}
private function onUpdateQuivTimer(event:Event):void
{
	if (_quivTimer.currentCount == 6)
	{
		_quivTimer.stop();
	}
}
private function onUpdateFrostDelayTimer(event:Event):void
{
	if (_frostTimerDelay.currentCount == 9)
	{
		_frostTimerDelay.stop();
	}
}
private function onUpdateTime(event:Event):void
{
	if (bladeSkel1 != null)
	{
```

```
                bladeSkel1.frozeSkel.visible = true;
                // stop timer when it reaches 0
                if (_freezeTimer.currentCount == 1)
                {
                        _freezeTimer.reset();
                        _freezeStatus = false;
                        bladeSkel1.frozeSkel.visible = true;
                }
                bladeSkel1.frozeSkel.visible = false;
        }
}
private function onUpdateHitTime(event:Event):void
{
        // this is incomplete for right now
        if (bladeSkel1 != null)
        {
                bladeSkel1.hurtSkel.visible = true;

                if (_enemyHitTimer.currentCount == 1)
                {
                        _enemyHitTimer.reset();
                        _enemyHit = false;
                        bladeSkel1.hurtSkel.visible = false;
                }
                bladeSkel1.hurtSkel.visible = false;
```

```
		}
}
// this code was made by Tyler Stansfield Jaggers 2013
// of overidon.com

private function onGnInvPressedLeft(event:Event)
{
		_gnInvNumber = _gnInvNumber - 1;
		if (_gnInvNumber == -1)
		{
			_gnInvNumber = 2;
			blinkInvBox2.visible = true;
			blinkInvBox1.visible = false;
			blinkInvBox0.visible = false;
		}
		else if (_gnInvNumber == 0)
		{
			blinkInvBox2.visible = false;
			blinkInvBox1.visible = false;
			blinkInvBox0.visible = true;
		}
		else if (_gnInvNumber == 1)
		{
			blinkInvBox2.visible = false;
			blinkInvBox1.visible = true;
```

```
                blinkInvBox0.visible = false;
        }

        else if (_gnInvNumber == 2)
        {
                blinkInvBox2.visible = true;
                blinkInvBox1.visible = false;
                blinkInvBox0.visible = false;
        }
}
private function onGnInvPressedRight(event:Event)
{
        _gnInvNumber = _gnInvNumber + 1;
        if (_gnInvNumber == 3)
        {
                _gnInvNumber = 0;
                blinkInvBox2.visible = false;
                blinkInvBox1.visible = false;
                blinkInvBox0.visible = true;
        }
        else if (_gnInvNumber == 0)
        {
                blinkInvBox2.visible = false;
                blinkInvBox1.visible = false;
                blinkInvBox0.visible = true;
```

```
		}
		else if (_gnInvNumber == 1)
		{
			blinkInvBox2.visible = false;
			blinkInvBox1.visible = true;
			blinkInvBox0.visible = false;
		}

		else if (_gnInvNumber == 2)
		{
			blinkInvBox2.visible = true;
			blinkInvBox1.visible = false;
			blinkInvBox0.visible = false;
		}
}
// begin item selection
private function onGnInvSelected(event:Event)
{
		if ((_gnInvNumber == 0) && (_gpHasBowInv == true))
		{
			quivEquip.visible = true;
			swordEquip.visible = false;
			spellEquip.visible = false;
			dispatchEvent(new Event("gnEquippedBow",true));
			_gpWeaponDamage = 50;
```

```
            _gpCriticalStrike = 50;


            this.inventoryMaster.quivAppearance.visible = true;


            this.inventoryMaster.unarmedAppearance.visible = false;
            this.inventoryMaster.swordAppearance.visible = false;
            this.inventoryMaster.frostAppearance.visible = false;
    }
    else if ((_gnInvNumber == 1) && (_gpHasSwordInv == true))
    {
            quivEquip.visible = false;
            swordEquip.visible = true;
            spellEquip.visible = false;
            dispatchEvent(new Event("gnEquippedSword",true));
            _gpWeaponDamage = 27;
            _gpCriticalStrike = 90;


            this.inventoryMaster.swordAppearance.visible = true;


            this.inventoryMaster.unarmedAppearance.visible = false;
            this.inventoryMaster.frostAppearance.visible = false;
            this.inventoryMaster.quivAppearance.visible = false;
    }
    else if ((_gnInvNumber == 2) && (_gpHasSpellInv ==  true))
    {
```

```
                quivEquip.visible = false;
                swordEquip.visible = false;
                spellEquip.visible = true;
                dispatchEvent(new Event("gnEquippedSpell",true));
                _gpWeaponDamage = 32;
                _gpCriticalStrike = 10;

                // add the sprite of the appearance
                this.inventoryMaster.frostAppearance.visible = true;

                this.inventoryMaster.unarmedAppearance.visible = false;
                this.inventoryMaster.swordAppearance.visible = false;
                this.inventoryMaster.quivAppearance.visible = false;

        }
}
// this CaveManager class is Copyright 2013 - Overidon.com - Tyler Stanfield Jaggers

private function onGnSwordHitFront(event:Event)
{
        bladeSkel1.y = bladeSkel1.y + 19;

}
private function onGnInvPressed(event:Event)
{
```

```
this.inventoryMaster.visible = !this.inventoryMaster.visible;


_gnInvStatus = !_gnInvStatus;


// invis some stuff
if (_gameVictoryStatus == false)
{
	if (spellPickup != null)
	{
		spellPickup.visible = !spellPickup.visible;
	}
	if (bowPickup != null)
	{
		bowPickup.visible = !bowPickup.visible;
	}
	if (swordPickup != null)
	{
		swordPickup.visible = !swordPickup.visible;
	}
}
if (bladeSkel1 != null)
{
	if (_gnInvStatus == true)
	{
		_tempVx = _skelVx;
```

```
            _tempVy = _skelVy;
            _tempX = bladeSkel1.x;
            _tempY = bladeSkel1.y;

            // begin disabling the skel
            _skelVx = 0;
            _skelVy = 0;
        }
        else
        {
            _skelVx = _tempVx;
            _skelVy = _tempVy;
        }
        if ((_tempVx * _tempVy <= 1) && (_gnInvStatus == false) )
        {
            _skelRandInv = Math.round(Math.random()) * 10;

            //begin movement shift
            if (_skelRandInv < 25)
            {
                _skelVx = 10;
                _skelVy = 0;
            }
            else if (50 > _skelRandInv > 25 )
            {
```

```
                    _skelVx = -10;
                    _skelVy = 0;
                }
                else if (75 > _skelRandInv > 50)
                {
                    _skelVy = 10;
                    _skelVx = 0;
                }
                else if (10 > _skelRandInv > 75)
                {
                    _skelVy = -10;
                    _skelVx = 0;
                }
            }
        }
    }
    private function onProjectileCreated(event:Event)
    {
        /// add  new projectiles to the array.
        // this should be empty for now
        _arrows -= 1;

        _projectiles.push(MovieClip(event.target));
    }
    private function onGnInvQuiv(event:Event)
```

```
{
		var bow:Bow = new Bow;
		this.inventoryMaster.inventorySlot0.addChild(bow);
		_gpWeaponDamage = 50;
		_gpCriticalStrike = 50;
}
private function onGnInvSword(event:Event)
{
		var swordPlaceholder:Sword = new Sword;
		this.inventoryMaster.inventorySlot1.addChild(swordPlaceholder);
		_gpWeaponDamage = 27;
		_gpCriticalStrike = 90;
}
private function onGnInvSpell(event:Event)
{
		var spellPickup1a:SpellPickup = new SpellPickup;
		this.inventoryMaster.inventorySlot2.addChild(spellPickup1a);
		_gpWeaponDamage = 32;
		_gpCriticalStrike = 10;
}
// begin the "ENTER FRAME" aspect of this code
// please note: this code might be controlled by a document class controller
// in future incarnations and evolutions of this game

private function onEnterFrame(event:Event):void
```

```
{
        // begin terminate congrats screen
        if (this.congrats.currentFrame >= 900)
        {
                this.congrats.stop();
        }
        // begin terminate tutorial
        if (this.tutorial != null)
        {
                if (this.tutorial.currentFrame == 284)
                {
                        this.tutorial.visible = false;
                        removeChild(tutorial);
                        tutorial = null;
                        myChannel.stop();
                        dispatchEvent(new Event("gameBegin",true));
                }
        }
        // begin skel hp drain notation
        if (_skelHp <= 360)
        {
                this.enemyHpBar10.visible = false;
        }
        else
        {
```

```
            this.enemyHpBar10.visible = true;
        }
        if (_skelHp <= 320)
        {
            this.enemyHpBar9.visible = false;
        }
        else
        {
            this.enemyHpBar9.visible = true;
        }
        if (_skelHp <= 280)
        {
            this.enemyHpBar8.visible = false;
        }
        else
        {
            this.enemyHpBar8.visible = true;
        }
        if (_skelHp <= 240)
        {
            this.enemyHpBar7.visible = false;
        }
        else
        {
            this.enemyHpBar7.visible = true;
```

```
}
if (_skelHp <= 200)
{
        this.enemyHpBar6.visible = false;
}
else
{
        this.enemyHpBar6.visible = true;
}
if (_skelHp <= 160)
{
        this.enemyHpBar5.visible = false;
}
else
{
        this.enemyHpBar5.visible = true;
}
if (_skelHp <= 120)
{
        this.enemyHpBar4.visible = false;
}
else
{
        this.enemyHpBar4.visible = true;
}
```

```
if (_skelHp <= 80)
{
	this.enemyHpBar3.visible = false;
}
else
{
	this.enemyHpBar3.visible = true;
}
if (_skelHp <= 40)
{
	this.enemyHpBar2.visible = false;
}
else
{
	this.enemyHpBar2.visible = true;
}
if (_skelHp <= 0)
{
	this.enemyHpBar1.visible = false;

	// dispatch an event which signals the end of the game
	dispatchEvent (new Event("gameVictory", true));
}
else
{
```

```
            this.enemyHpBar1.visible = true;
}
// begin gp hp drain notation
if (_gpHp <= 90)
{
            this.gnHpBar10.visible = false;
}
else
{
            this.gnHpBar10.visible = true;
}
if (_gpHp <= 80)
{
            this.gnHpBar9.visible = false;
}
else
{
            this.gnHpBar9.visible = true;
}
if (_gpHp <= 70)
{
            this.gnHpBar8.visible = false;
}
else
{
```

```
            this.gnHpBar8.visible = true;
}
if (_gpHp <= 60)
{
            this.gnHpBar7.visible = false;
}
else
{
            this.gnHpBar7.visible = true;
}
if (_gpHp <= 50)
{
            this.gnHpBar6.visible = false;
}
else
{
            this.gnHpBar6.visible = true;
}
if (_gpHp <= 40)
{
            this.gnHpBar5.visible = false;
}
else
{
            this.gnHpBar5.visible = true;
```

```
}
if (_gpHp <= 30)
{
        this.gnHpBar4.visible = false;
}
else
{
        this.gnHpBar4.visible = true;
}
if (_gpHp <= 20)
{
        this.gnHpBar3.visible = false;
}
else
{
        this.gnHpBar3.visible = true;
}
if (_gpHp <= 10)
{
        this.gnHpBar2.visible = false;
}
else
{
        this.gnHpBar2.visible = true;
}
```

```
if (_gpHp <= 0)
{
	this.gnHpBar1.visible = false;
	this.gameOver.visible = true;
	gnomePrince.visible = false;
	gnomePrince.x = 650;

	// dispatch the game over event to all classes
	dispatchEvent(new Event("gameOver", true));
}
else
{
	this.gnHpBar1.visible = true;
}

// attempt to create a system to destroy projectiles if the inventory is up
// this gets rid of the "blaster master" grenade-style bug :D

if (this.inventoryMaster.visible == true)
{
	for (var i:int = 0; i < _projectiles.length; i++)
	{
		switch (_projectiles[i].projectileType)
		{
			case "QUIV" :
```

```
                    // remove that projectile from the stage
                    removeChild(_projectiles[i]);

                    // remove the projectile from the array
                    _projectiles.splice(i,1);

                    break;

                case "SPELL" :
                    // remove that projectile from the stage
                    removeChild(_projectiles[i]);

                    // remove the projectile from the array
                    _projectiles.splice(i,1);

                    break;
            }
        }
    }
    if ((_skelHp <= 0) && (bladeSkel1 != null))
    {
        _deathLocationX = bladeSkel1.x;
        _deathLocationY = bladeSkel1.y;

        removeChild(bladeSkel1);
```

```
            bladeSkel1 = null;


            deathAnimation = new DeathAnimation();
            this.addChild(deathAnimation);


            deathAnimation.x = _deathLocationX;
            deathAnimation.y = _deathLocationY;


}
if  ((deathAnimation != null) && (deathAnimation.currentFrame == 7) )
{
            removeChild(deathAnimation);
            deathAnimation = null;
}
// this is the "Freeze Timer" or froze spell effect area
// this didn't work at first and crashed my as3 prog


if ((_freezeStatus == true) && (bladeSkel1 != null))
{
            _skelVx = (0.9) * _skelVx;
            _skelVy = (0.9) * _skelVy;
}


// begin the test for stat checks
```

```
this.inventoryMaster.hpBox.text = _gpHp + "/100";
this.inventoryMaster.weaponDamageBox.text = String(_gpWeaponDamage);
this.inventoryMaster.criticalStrikeBox.text = String(_gpCriticalStrike) + "%";


// begin the check for if the gnome accidentaly touches the skeleton
if (bladeSkel1 != null)
{
	if (gnomePrince.gnHitBox.hitTestObject(bladeSkel1) )
	{
		if (gnomePrince.pubGnVx > 0)
		{
			gnomePrince.x = gnomePrince.x - 30;
		}
		else if (gnomePrince.pubGnVx < 0)
		{
			gnomePrince.x = gnomePrince.x + 30;
		}
		if (gnomePrince.pubGnVy > 0)
		{
			gnomePrince.y = gnomePrince.y - 30;
		}
		else if (gnomePrince.pubGnVy < 0)
		{
			gnomePrince.y = gnomePrince.y + 30;
		}
```

```
                _gpHp = _gpHp - 1;
        }
}

// begin the skeleton hit the gnome attack
// this is due to the player simply not getting out of the way fast
//enough or maybe he got overwhelmed

if (bladeSkel1 != null)
{
        if (bladeSkel1.hitTestObject(gnomePrince.gnHitBox))
        {
                if (_skelVx > 0)
                {
                        gnomePrince.x = gnomePrince.x + 20;
                }
                else if (_skelVx < 0)
                {
                        gnomePrince.x = gnomePrince.x - 20;
                }
                else if (_skelVy > 0)
                {
                        gnomePrince.y = gnomePrince.y + 20;
                }
                else if (_skelVy < 0)
```

```
                {
                        gnomePrince.y = gnomePrince.y - 20;
                }
                _gpHp = _gpHp - 1;
        }
}

// begin the skeleton getting hit by the sword attacks
if (bladeSkel1 != null)
{
        if (gnomePrince.gpArFrontAttack.attackHitBoxFront != null)
        {
                if
(gnomePrince.gpArFrontAttack.attackHitBoxFront.hitTestObject(bladeSkel1) )
                {
                        _critRand = Math.random() * 100;

                        // in this area you can affect the HP of the enemy
                        if (_critRand >=10 )
                        {
                                _skelHp = _skelHp - ( (_gpWeaponDamage) * 2);
                        }
                        else
                        {
                                _skelHp = _skelHp - ( (_gpWeaponDamage));
                        }
```

```
                                        // affect movement
                                        _enemyHitTimer.start();
                                        bladeSkel1.hurtSkel.visible = true;
                                        bladeSkel1.y = bladeSkel1.y + 20;

                                        // create hit sound
                                        if (_soundActive == true)
                                        {
                                                enemyHit = new EnemyHit();
                                                enemyHit.play();

                                        }

                                }
                        }
                        if (gnomePrince.gpArBackAttack.attackHitBoxBack != null)
                        {
                                if
(gnomePrince.gpArBackAttack.attackHitBoxBack.hitTestObject(bladeSkel1) )
                                {
                                        _critRand = Math.random() * 100;

                                        // in this area you can affect the HP of the enemy
                                        if (_critRand >=10 )
                                        {
                                                _skelHp = _skelHp - ( (_gpWeaponDamage) * 2);
```

```
                    }
                    else
                    {
                        _skelHp = _skelHp - ( (_gpWeaponDamage));
                    }
                    // affect movement
                    _enemyHitTimer.start();
                    bladeSkel1.hurtSkel.visible = true;
                    bladeSkel1.y = bladeSkel1.y - 20;

                    // create hit sound
                    if (_soundActive == true)
                    {
                        enemyHit = new EnemyHit();
                        enemyHit.play();

                    }
                }
            }
            if (gnomePrince.gpArLeftAttack.attackHitBoxLeft != null)
            {
                if
(gnomePrince.gpArLeftAttack.attackHitBoxLeft.hitTestObject(bladeSkel1) )
                {
                    _critRand = Math.random() * 100;
```

```
                    // in this area you can affect the HP of the enemy
                    if (_critRand >=10 )
                    {
                            _skelHp = _skelHp - ( (_gpWeaponDamage) * 2);
                    }
                    else
                    {
                            _skelHp = _skelHp - ( (_gpWeaponDamage));
                    }
                    // affect movement
                    _enemyHitTimer.start();
                    bladeSkel1.hurtSkel.visible = true;
                    bladeSkel1.x = bladeSkel1.x - 20;

                    // create hit sound
                    if (_soundActive == true)
                    {
                            enemyHit = new EnemyHit();
                            enemyHit.play();

                    }
            }
    }
    if (gnomePrince.gpArRightAttack.attackHitBoxRight != null)
    {
```

```
                                        if
(gnomePrince.gpArRightAttack.attackHitBoxRight.hitTestObject(bladeSkel1) )
                                        {
                                                _critRand = Math.random() * 100;

                                                // in this area you can affect the HP of the enemy
                                                if (_critRand >=10 )
                                                {
                                                        _skelHp = _skelHp - ( (_gpWeaponDamage) * 2);
                                                }
                                                else
                                                {
                                                        _skelHp = _skelHp - ( (_gpWeaponDamage));
                                                }
                                                // affect movement
                                                _enemyHitTimer.start();
                                                bladeSkel1.hurtSkel.visible = true;
                                                bladeSkel1.x = bladeSkel1.x + 20;

                                                // create hit sound
                                                if (_soundActive == true)
                                                {
                                                        enemyHit = new EnemyHit();
                                                        enemyHit.play();

                                                }
```

```
                    }

                }

            }

            // begin the test bladeskel1 ai

            if (bladeSkel1 != null)

            {

                if ((southBorder.hitTestObject(bladeSkel1)) ||
(westBorder.hitTestObject(bladeSkel1)) || (eastBorder.hitTestObject(bladeSkel1)) ||
(northBorder.hitTestObject(bladeSkel1)))

                {

                    if (_skelVx != 0)

                    {

                        _skelVx = 0;

                        if (bladeSkel1.x >= 350)

                        {

                            bladeSkel1.x = 488;

                        }

                        else

                        {

                            bladeSkel1.x = 64;

                        }

                        _binaryRand = Math.round(Math.random());


                        if (_binaryRand == 1)

                        {

                            _skelVy = -6;
```

```
		}
		else
		{
			_skelVy = 6;
		}
	}
	else if (_skelVy != 0)
	{
		_skelVy = 0
		_binaryRand = Math.round(Math.random());

		if (bladeSkel1.y >= 272)
		{
			bladeSkel1.y = 338;
		}
		else
		{
			bladeSkel1.y = 73;
		}

		if (_binaryRand == 1)
		{
			_skelVx = -6;
		}
		else
```

```
                        {
                                _skelVx = 6;
                        }
                }
        }
}

if (southBorder.hitTestObject(gnomePrince.gnHitBox))
{
        dispatchEvent(new Event("gnWallHitSouth",true));
}
if (eastBorder.hitTestObject(gnomePrince.gnHitBox))
{
        dispatchEvent(new Event("gnWallHitEast",true));
}
if (westBorder.hitTestObject(gnomePrince.gnHitBox))
{
        dispatchEvent(new Event("gnWallHitWest",true));
}
if (northBorder.hitTestObject(gnomePrince.gnHitBox))
{
        dispatchEvent(new Event("gnWallHitNorth",true));
}
        // begin the test bladeSkel1 movement
        if (bladeSkel1 != null)
```

```
{
        _skelTrav = _skelTrav + 1;

        if (_skelTrav >= 70)
        {
                if (_skelVx != 0)
                {
                        _skelVx = 0;

                        if(bladeSkel1.y > 350)
                        {
                                _skelVy = -9;
                        }
                        else
                        {
                                _skelVy = 9;
                        }
                }
                else if (_skelVy != 0 )
                {
                        _skelVy = 0;

                        if (bladeSkel1.x > 350)
                        {
                                _skelVx = -9;
```

```
                                        }
                                        else
                                        {
                                                _skelVx = 9;
                                        }
                                }
                                _skelTrav = 0;
                        }
                        bladeSkel1.x = bladeSkel1.x + _skelVx;
                        bladeSkel1.y = bladeSkel1.y + _skelVy;
                }
        if (bladeSkel1 == null)
        {
                for (var i:int = 0; i < _projectiles.length; i++)
                {
                        switch (_projectiles[i].projectileType)
                        {
                                case "QUIV" :

                                        if (_projectiles[i].hitTestObject(westBorder) ||
_projectiles[i].hitTestObject(eastBorder) || _projectiles[i].hitTestObject(northBorder) ||
_projectiles[i].hitTestObject(southBorder) )
                                        {
                                                // remove that projectile from the stage
                                                removeChild(_projectiles[i]);
```

```
                                        // remove the projectile from the array
                                        _projectiles.splice(i,1);
                                    }
                                    break;

                                case "SPELL" :

                                    if (_projectiles[i].hitTestObject(westBorder) ||
_projectiles[i].hitTestObject(eastBorder) || _projectiles[i].hitTestObject(northBorder) ||
_projectiles[i].hitTestObject(southBorder) )
                                    {
                                        // remove that projectile from the stage
                                        removeChild(_projectiles[i]);

                                        // remove the projectile from the array
                                        _projectiles.splice(i,1);
                                    }
                                    break;
                            }
                        }
                    }
                    if (bladeSkel1 != null)
                    {
                        for (var i:int = 0; i < _projectiles.length; i++)
                        {
                            switch (_projectiles[i].projectileType)
```

```
{
                    case "QUIV" :

                        // check for a collision with the enemy
bladeSkel1

                        if (bladeSkel1.hitTestObject(_projectiles[i]))
                        {
                            bladeSkel1.x = bladeSkel1.x +
(_projectiles[i].projectileVx * 3);

                            bladeSkel1.y = bladeSkel1.y +
(_projectiles[i].projectileVy * 3);

                            // remove that projectile from the stage
                            removeChild(_projectiles[i]);

                            // remove the projectile from the array
                            _projectiles.splice(i,1);

                            // subtract 1 from the counter to
compensate
                            // for the removed array element
                            i--;

                            // check the crit status
                            _critRand = Math.random() * 100;
```

```
                    // in this area you can affect the HP of
the enemy
                    if (_critRand >=51 )
                    {
                        _skelHp = _skelHp - (
(_gpWeaponDamage) * 2);
                    }
                    else
                    {
                        _skelHp = _skelHp - (
(_gpWeaponDamage));
                    }
                    _enemyHitTimer.start();
                    bladeSkel1.hurtSkel.visible = true;
                    // enemy hit sound
                    if (_soundActive == true)
                    {
                        enemyHit = new EnemyHit();
                        enemyHit.play();

                    }
                }
                // check for collision with the wall
                else if (_projectiles[i].hitTestObject(westBorder)
|| _projectiles[i].hitTestObject(eastBorder) || _projectiles[i].hitTestObject(northBorder) ||
_projectiles[i].hitTestObject(southBorder) )
                {
```

```
                    // remove that projectile from the stage
                    removeChild(_projectiles[i]);

                    // remove the projectile from the array
                    _projectiles.splice(i,1);
                }
                break;

            case "SPELL" :

                // check for a collision with the enemy
bladeSkel1
                if (bladeSkel1.hitTestObject(_projectiles[i]))
                {

                    // remove that projectile from the stage
                    removeChild(_projectiles[i]);

                    // remove the projectile from the array
                    _projectiles.splice(i,1);

                    // subtract 1 from the counter to
compensate
                    // for the removed array element

                    i--;
```

```
// check for crits
_critRand = Math.random() * 100;

// in this area you can affect the HP of
the enemy
if (_critRand >= 87)
{
        _skelHp = _skelHp - (
(_gpWeaponDamage) * 2);
}
else
{
        _skelHp = _skelHp - (
(_gpWeaponDamage));
}

// in this area I'm considering freezing
the skeleton
// this would make sense because I
used a "frost" type of color scheme
// for Gnome Prince associated with the
scroll
_freezeTimer.start();
bladeSkel1.frozeSkel.visible = true;
_freezeStatus = true;

// add frost hit sound
if (_soundActive == true)
```

```
                                        {
                                                frostHit = new FrostHit();
                                                frostHit.play();

                                        }
                                }
                                else if (_projectiles[i].hitTestObject(westBorder)
|| _projectiles[i].hitTestObject(eastBorder) || _projectiles[i].hitTestObject(northBorder) ||
_projectiles[i].hitTestObject(southBorder) )
                                {
                                        // remove that projectile from the stage
                                        removeChild(_projectiles[i]);

                                        // remove the projectile from the array
                                        _projectiles.splice(i,1);
                                }
                                break;
                        }
                }
        }

        if ((bowPickup != null) && (gnomePrince.gnHitBox.hitTestObject(bowPickup)))
        {
                _gpHasBowInv = true;
                removeChild(bowPickup);
                bowPickup = null;
                _arrows += 50;
```

```
                    dispatchEvent(new Event("gnInvQuiv",true));
                    quivEquip.visible = true;
                    swordEquip.visible = false;
                    spellEquip.visible = false;
                    dispatchEvent(new Event("gnEquippedBow",true));

                    this.inventoryMaster.quivAppearance.visible = true;

                    this.inventoryMaster.unarmedAppearance.visible = false;
                    this.inventoryMaster.swordAppearance.visible = false;
                    this.inventoryMaster.frostAppearance.visible = false;

                }

                if ((swordPickup != null) &&
(gnomePrince.gnHitBox.hitTestObject(swordPickup)))
                {
                    _gpHasSwordInv = true;
                    removeChild(swordPickup);
                    swordPickup = null;

                    // now the dispatch
                    dispatchEvent(new Event("gnInvSword" , true));

                    quivEquip.visible = false;
                    swordEquip.visible = true;
```

```
            spellEquip.visible = false;
            dispatchEvent(new Event("gnEquippedSword",true));

            this.inventoryMaster.swordAppearance.visible = true;

            this.inventoryMaster.unarmedAppearance.visible = false;
            this.inventoryMaster.quivAppearance.visible = false;
            this.inventoryMaster.frostAppearance.visible = false;
        }

        if ((spellPickup != null) && (gnomePrince.gnHitBox.hitTestObject(spellPickup)))
        {
            _gpHasSpellInv = true;
            removeChild(spellPickup);
            spellPickup = null;

            //dispatch the spellPickup event
            dispatchEvent(new Event("gnInvSpell",true));
            quivEquip.visible = false;
            swordEquip.visible = false;
            spellEquip.visible = true;
            dispatchEvent(new Event("gnEquippedSpell",true));

            this.inventoryMaster.frostAppearance.visible = true;
```

```
                this.inventoryMaster.unarmedAppearance.visible = false;
                this.inventoryMaster.swordAppearance.visible = false;
                this.inventoryMaster.quivAppearance.visible = false;
            }

            if (bladeSkel1 != null)
            {
                if (this.inventoryMaster.visible == true)
                {
                    bladeSkel1.visible = false;
                }
                else
                {
                    bladeSkel1.visible = true;
                }
            }
        }
    }
}
```

SECTION 2 – GPUNARMED CLASS

```
package
{
```

```
import flash.display.MovieClip;
import flash.events.KeyboardEvent;
import flash.ui.Keyboard;
import flash.events.Event;

// timer event preparation for quiver delay

import flash.events.TimerEvent;
import flash.utils.Timer;

public class GpUnarmed extends MovieClip
{
	// Begin Code
	// This class file was created by Tyler Stansfield Jaggers
	// copyright overidon.com March 8th, 2013

	private var _vx:int;
	private var _vy:int;
	private var _gnDirection:String;
	private var _shotFired:Boolean;
	private var _gnInventory:String;

	// create a sword attacking variable that prevents sword auto-fire
	private var _swordAttacking:Boolean;
```

```
// create variable that prevents arrows from being fired
private var _quiverDrawnOk:Boolean;
private var _frostDrawnOk:Boolean;
private var _swordDrawnOk:Boolean;

// create variable to hold if music and sound is on
private var _soundActive:Boolean;

// duplicate gameOver status for the gp class
private var _gameOverStatus:Boolean;

var gameMode:String;
var pubGnVx:int;
var pubGnVy:int;

var frostCast:FrostCast;
var quiverPull:QuiverPull;
var swordThrust:SwordThrust;

public function GpUnarmed()
{
        // initialize variables
        gameMode = "ACTION";
        _vx = 0;
        _vy = 0;
```

```
_gnInventory = "NONE";


_gameOverStatus = false;


_shotFired = false;
_swordAttacking = false;


_quiverDrawnOk = false;
_frostDrawnOk = false;
_swordDrawnOk = false;


// set sound to true
_soundActive = true;



// make the attack mode for left invisible to try and see if
// I can save some copy paste


// I'm probably going to have to retool for a hit-box.
// the attacking sprite is too wide to continue using
// basic hitTestObject without subObjects


this.gpArLeftAttack.visible = false;
this.gpArLeftAttack.stop();
this.gpArLeftAttack.frame = 1;
```

```
this.gpArRightAttack.visible = false;
this.gpArRightAttack.stop();
this.gpArRightAttack.frame = 1;
this.gpArBackAttack.visible = false;
this.gpArBackAttack.stop();
this.gpArBackAttack.frame = 1;
this.gpArFrontAttack.visible = false;
this.gpArFrontAttack.stop();
this.gpArFrontAttack.frame = 1;

// add stage event listeners for the Gnome Prince
// specifically key presses and "enter frames"
// NOTE: The enter frame aspect might be
// re-routed to a document class controller
// for overlapping consistency
stage.addEventListener(KeyboardEvent.KEY_DOWN, onKeyDown);
stage.addEventListener(KeyboardEvent.KEY_UP, onKeyUp);
stage.addEventListener("gnInvQuiv", onGnInvQuiv);
addEventListener(Event.ENTER_FRAME, onEnterFrame);
stage.addEventListener("gnInvSword", onGnInvSword);
stage.addEventListener("gnInvSpell", onGnInvSpell);
stage.addEventListener("gnEquippedBow", onGnEquippedBow);
stage.addEventListener("gnEquippedSword", onGnEquippedSword);
stage.addEventListener("gnEquippedSpell", onGnEquippedSpell);
stage.addEventListener("gnWallHitSouth", onGnWallHitSouth);
```

```
stage.addEventListener("gnWallHitEast", onGnWallHitEast);
stage.addEventListener("gnWallHitWest", onGnWallHitWest);
stage.addEventListener("gnWallHitNorth", onGnWallHitNorth);

// add an event listener for the quiver drawn issue for delay etc
stage.addEventListener("quivDrawnOk", onQuivDrawnOk);
stage.addEventListener("quivDrawnNotReady", onQuivDrawnNotReady);

//add event listers for the frost delay issue
stage.addEventListener("frostDrawnOk", onFrostDrawnOk);
stage.addEventListener("frostDrawnNotReady", onFrostDrawnNotReady);

// add event listeners for the sword delay issue
stage.addEventListener("swordDrawnOk", onSwordDrawnOk);
stage.addEventListener("swordDrawnNotReady", onSwordDrawnNotReady);

// add event listeners for the button presses
stage.addEventListener("upButtonClicked", onUpButtonClicked);
stage.addEventListener("upButtonReleased", onUpButtonReleased);
stage.addEventListener("downButtonClicked", onDownButtonClicked);
stage.addEventListener("downButtonReleased", onDownButtonReleased);
stage.addEventListener("leftButtonClicked", onLeftButtonClicked);
stage.addEventListener("leftButtonReleased", onLeftButtonReleased);
stage.addEventListener("rightButtonClicked", onRightButtonClicked);
stage.addEventListener("rightButtonReleased", onRightButtonReleased);
```

```
                stage.addEventListener("spaceBarButtonClicked", onSpaceBarButtonClicked);
                stage.addEventListener("spaceBarButtonReleased",
onSpaceBarButtonReleased);

                stage.addEventListener("inventoryButtonClicked", onInventoryButtonClicked);
                stage.addEventListener("inventoryButtonReleased",
onInventoryButtonReleased);

                // add an event listener to see if sound is active
                stage.addEventListener("soundToggle", onSoundToggle);

                // add an event listener for gameOver status change
                stage.addEventListener("gameOver", onGameOver);
                // add an event listener for gameVictory Status change
                stage.addEventListener("gameVictory", onGameVictory);
                // add an event listener for the sound enabling
                stage.addEventListener("soundEnabled", onSoundEnabled);
        }
        private function onSoundEnabled(event:Event):void
        {
                _soundActive = true;
        }
        private function onGameOver(event:Event):void
        {
                _gameOverStatus = true;
```

```
            _soundActive = false;
            _gnInventory = "NONE";
        }
        private function onGameVictory(event:Event):void
        {
            _gameOverStatus = true;
            _soundActive = false;
            _gnInventory = "NONE";
        }
        private function onSoundToggle(event:Event)
        {
            _soundActive = !_soundActive;
        }
        private function onSwordDrawnOk(event:Event)
        {
            _swordDrawnOk = true;
        }
        private function onSwordDrawnNotReady(event:Event)
        {
            _swordDrawnOk = false;
        }
        private function onFrostDrawnOk(event:Event)
        {
            _frostDrawnOk = true;
        }
```

```
private function onFrostDrawnNotReady(event:Event)
{
	_frostDrawnOk = false;
}
private function onQuivDrawnOk(event:Event)
{
	_quiverDrawnOk = true;
}
private function onQuivDrawnNotReady(event:Event)
{
	_quiverDrawnOk = false;
}
// begin the listen event for the inventory change
// these are like "hot pickups" for events!
private function onGnInvQuiv(event:Event)
{
	_gnInventory = "QUIV";
}
private function onGnInvSword(event:Event)
{
	_gnInventory = "SWORD";
}
private function onGnInvSpell(event:Event)
{
	_gnInventory = "SPELL";
```

```
}
// begin the event listeners for equippment changes from the inventory screen
private function onGnEquippedBow(event:Event)
{
	_gnInventory = "QUIV";
}
private function onGnEquippedSword(event:Event)
{
	_gnInventory = "SWORD";
}
private function onGnEquippedSpell(event:Event)
{
	_gnInventory = "SPELL";
}

// Begin the collision detection for the gnome and the walls

private function onGnWallHitSouth(event:Event)
{
	this.y = this.y + -(_vy);

	this.y = 344;
}
private function onGnWallHitWest(event:Event)
{
```

```
            this.x = this.x + -(_vx);
            this.x = 56;
}
private function onGnWallHitEast(event:Event)
{
            this.x = this.x + -(_vx);
            this.x = 497;
}
private function onGnWallHitNorth(event:Event)
{
            this.y = this.y + -(_vx);
            this.y = 66;
}

// begin the mouse button keyboard dupes

private function onUpButtonClicked(event:Event)
{
            if (gameMode == "ACTION")
            {
                        _vy = -5;
                        _vx = 0;
                        _gnDirection = "back";
                        //test public variable
                        pubGnVx = _vx;
```

```
                pubGnVy = _vy;
            }
        }
        private function onUpButtonReleased(event:Event)
        {
            if (gameMode == "ACTION")
            {
                _vy = 0;
                _vx = 0;
                pubGnVy = _vy;
            }
        }
        private function onDownButtonClicked(event:Event)
        {
            if (gameMode == "ACTION")
            {
                _vy = 5;
                _vx = 0;
                _gnDirection = "front";
                //test public variable
                pubGnVx = _vx;
                pubGnVy = _vy;
            }
        }
        private function onDownButtonReleased(event:Event)
```

```
{
        if (gameMode == "ACTION")
        {
                _vy = 0;
                _vx = 0;

                pubGnVy = _vy;
        }
}
private function onLeftButtonClicked(event:Event)
{
        if (gameMode == "ACTION")
        {
                _vx = -5;
                _vy = 0;
                _gnDirection = "left";
                dispatchEvent(new Event("gnDirection", true));
                //test public variable
                pubGnVx = _vx;
                pubGnVy = _vy;
        }
        else if (gameMode == "INVENTORY")
        {
                dispatchEvent(new Event("gnInvPressedLeft",true));
        }
```

```
}
private function onLeftButtonReleased(event:Event)
{
	if (gameMode == "ACTION")
	{
		_vy = 0;
		_vx = 0;
		// test pub variable
		pubGnVx = _vx;
	}
}
private function onRightButtonClicked(event:Event)
{
	if (gameMode == "ACTION")
	{
		_vx = 5;
		_vy = 0;
		_gnDirection = "right";
		//test public variable
		pubGnVx = _vx;
		pubGnVy = _vy;
	}
	else if (gameMode == "INVENTORY")
	{
		dispatchEvent(new Event("gnInvPressedRight",true));
```

```
		}
	}
	private function onRightButtonReleased(event:Event)
	{
		if (gameMode == "ACTION")
		{
			_vy = 0;
			_vx = 0;
			// test pub variable
			pubGnVx = _vx;
		}
	}
	private function onInventoryButtonClicked(event:Event)
	{
		if (gameMode == "ACTION")
		{
			dispatchEvent(new Event("gnInvPressed",true));
			gameMode = "INVENTORY";

		}
		else if (gameMode == "INVENTORY")
		{
			dispatchEvent(new Event("gnInvPressed",true));
			gameMode = "ACTION";

		}
```

```
}
private function onInventoryButtonReleased(event:Event)
{
	trace(gameMode);
}
private function onSpaceBarButtonReleased(event:Event)
{
	_shotFired = false;
}
private function onSpaceBarButtonClicked(event:Event)
{
	if (_gameOverStatus == false)
	{
		if (gameMode == "INVENTORY")
		{
			dispatchEvent(new Event("gnInvSelected",true));
		}
		else if (gameMode == "ACTION")
		{
			if ((! _shotFired) && (_gnInventory == "QUIV"))
			{
				dispatchEvent(new Event("quivDelayCheck",true));

				if (_quiverDrawnOk == true)
				{
```

```
                    shootProjectile();
                    _shotFired = true;

                    // play sound
                    if (_soundActive == true)
                    {
                        quiverPull = new QuiverPull();
                        quiverPull.play();

                    }

            }
    }
    else if ((! _shotFired) && (_gnInventory == "SPELL"))
    {
            dispatchEvent(new Event("frostDelayCheck",true));

            if (_frostDrawnOk == true)
            {
                    shootProjectile();
                    _shotFired = true;

                    // play sound
                    if (_soundActive == true)
                    {
                        frostCast = new FrostCast();
```

```
                    frostCast.play();

                }
            }
        }
        else if (_gnInventory == "SWORD")
        {
            dispatchEvent(new Event("swordDelayCheck", true));

            if (_swordDrawnOk == false)
            {
                trace("sword not ready")
            }
            else if (_swordDrawnOk == true)
            {
                if (_gnDirection == "left")
                {
                    this.gpArLeftAttack.frame = 1;
                    this.gpArLeftAttack.visible = true;
                    this.gpArLeftAttack.play();

                    // invis everyone else
                    this.gpArFrontAttack.visible = false;
                    this.gpArBackAttack.visible = false;

                    this.gpArRightAttack.visible = false;
```

```
            this.gpUnFrontQuiv.visible = false;
            this.gpUnBackQuiv.visible = false;
            this.gpUnLeftQuiv.visible = false;
            this.gpUnRightQuiv.visible = false;
            this.gpArFront.visible = false;
            this.gpArBack.visible = false;
            this.gpArLeft.visible = false;
            this.gpArRight.visible = false;
            this.gpUnFrontFr.visible = false;
            this.gpUnBackFr.visible = false;
            this.gpUnLeftFr.visible = false;
            this.gpUnRightFr.visible = false;
        }
        if (_gnDirection == "right")
        {
            this.gpArRightAttack.frame = 1;
            this.gpArRightAttack.visible = true;
            this.gpArRightAttack.play();

            // invis everyone else
            this.gpArFrontAttack.visible = false;
            this.gpArBackAttack.visible = false;
            this.gpArLeftAttack.visible = false;
            this.gpUnFrontQuiv.visible = false;
            this.gpUnBackQuiv.visible = false;
```

```
            this.gpUnLeftQuiv.visible = false;
            this.gpUnRightQuiv.visible = false;
            this.gpArFront.visible = false;
            this.gpArBack.visible = false;
            this.gpArLeft.visible = false;
            this.gpArRight.visible = false;
            this.gpUnFrontFr.visible = false;
            this.gpUnBackFr.visible = false;
            this.gpUnLeftFr.visible = false;
            this.gpUnRightFr.visible = false;
        }
        if (_gnDirection == "back")
        {
            this.gpArBackAttack.frame = 1;
            this.gpArBackAttack.visible = true;
            this.gpArBackAttack.play();

            // invis everyone else
            this.gpArFrontAttack.visible = false;
            this.gpArRightAttack.visible = false;
            this.gpArLeftAttack.visible = false;
            this.gpUnFrontQuiv.visible = false;
            this.gpUnBackQuiv.visible = false;
            this.gpUnLeftQuiv.visible = false;
            this.gpUnRightQuiv.visible = false;
```

```
            this.gpArFront.visible = false;
            this.gpArBack.visible = false;
            this.gpArLeft.visible = false;
            this.gpArRight.visible = false;
            this.gpUnFrontFr.visible = false;
            this.gpUnBackFr.visible = false;
            this.gpUnLeftFr.visible = false;
            this.gpUnRightFr.visible = false;

        }
        if (_gnDirection == "front")
        {
            this.gpArFrontAttack.frame = 1;
            this.gpArFrontAttack.visible = true;
            this.gpArFrontAttack.play();

            // invis everyone else
            this.gpArBackAttack.visible = false;
            this.gpArRightAttack.visible = false;
            this.gpArLeftAttack.visible = false;
            this.gpUnFrontQuiv.visible = false;
            this.gpUnBackQuiv.visible = false;
            this.gpUnLeftQuiv.visible = false;
            this.gpUnRightQuiv.visible = false;
            this.gpArFront.visible = false;
```

```
                                        this.gpArBack.visible = false;
                                        this.gpArLeft.visible = false;
                                        this.gpArRight.visible = false;
                                        this.gpUnFrontFr.visible = false;
                                        this.gpUnBackFr.visible = false;
                                        this.gpUnLeftFr.visible = false;
                                        this.gpUnRightFr.visible = false;
                                        // end of the "spacebar" attack in the
actionmode
                                        // this correlates to the button with
mouse only
                                    }
                                    // play sound
                                    if (_soundActive == true)
                                    {
                                        swordThrust = new SwordThrust();
                                        swordThrust.play();

                                    }
                                }
                            }
                        }
                    }
                }


                // begin the "KEY PRESS DOWN function"
```

```
private function onKeyDown(event:KeyboardEvent):void
{
	if((event.keyCode == Keyboard.LEFT)	&& (gameMode == "ACTION") )

	{
		_vx = -5;
		_vy = 0;
		_gnDirection = "left";
		dispatchEvent(new Event("gnDirection", true));

		//test public variable
		pubGnVx = _vx;
		pubGnVy = _vy;

	}
	else if ((event.keyCode == Keyboard.RIGHT)	&& (gameMode == "ACTION") )
	{
		_vx = 5;
		_vy = 0;
		_gnDirection = "right";
		//test public variable
		pubGnVx = _vx;
		pubGnVy = _vy;
	}
	else if ((event.keyCode == Keyboard.UP)&& (gameMode == "ACTION") )
	{
```

```
            _vy = -5;
            _vx = 0;
            _gnDirection = "back";
            //test public variable
            pubGnVx = _vx;
            pubGnVy = _vy;
    }
    else if ((event.keyCode == Keyboard.DOWN) && (gameMode == "ACTION") )
    {
            _vy = 5;
            _vx = 0;
            _gnDirection = "front";
            //test public variable
            pubGnVx = _vx;
            pubGnVy = _vy;
    }
    else if ((event.keyCode == Keyboard.SPACE)        && (gameMode == "ACTION") )
    {
            if ((! _shotFired) && (_gnInventory == "QUIV"))
            {
                    dispatchEvent(new Event("quivDelayCheck",true));

                    if (_quiverDrawnOk == true)
                    {
                            shootProjectile();
```

```
                    _shotFired = true;


                    // play sound
                    if (_soundActive == true)
                    {
                            quiverPull = new QuiverPull();
                            quiverPull.play();
                    }
            }
    }
    else if ((! _shotFired) && (_gnInventory == "SPELL"))
    {
            dispatchEvent(new Event("frostDelayCheck",true));


            if (_frostDrawnOk == true)
            {
                    shootProjectile();
                    _shotFired = true;


                    // play sound
                    if (_soundActive == true)
                    {
                            frostCast = new FrostCast();
                            frostCast.play();


                    }
```

```
            }
}
else if (_gnInventory == "SWORD")
{
            dispatchEvent(new Event("swordDelayCheck", true));

            if (_swordDrawnOk == false)
            {
                        trace("sword not ready")
            }
            else if (_swordDrawnOk == true)
            {
                        if (_gnDirection == "left")
                        {
                                    this.gpArLeftAttack.frame = 1;
                                    this.gpArLeftAttack.visible = true;
                                    this.gpArLeftAttack.play();

                                    // invis everyone else
                                    this.gpArFrontAttack.visible = false;
                                    this.gpArBackAttack.visible = false;

                                    this.gpArRightAttack.visible = false;
                                    this.gpUnFrontQuiv.visible = false;
                                    this.gpUnBackQuiv.visible = false;
                                    this.gpUnLeftQuiv.visible = false;
```

```
            this.gpUnRightQuiv.visible = false;
            this.gpArFront.visible = false;
            this.gpArBack.visible = false;
            this.gpArLeft.visible = false;
            this.gpArRight.visible = false;
            this.gpUnFrontFr.visible = false;
            this.gpUnBackFr.visible = false;
            this.gpUnLeftFr.visible = false;
            this.gpUnRightFr.visible = false;
        }
        if (_gnDirection == "right")
        {
            this.gpArRightAttack.frame = 1;
            this.gpArRightAttack.visible = true;
            this.gpArRightAttack.play();

            // invis everyone else
            this.gpArFrontAttack.visible = false;
            this.gpArBackAttack.visible = false;
            this.gpArLeftAttack.visible = false;
            this.gpUnFrontQuiv.visible = false;
            this.gpUnBackQuiv.visible = false;
            this.gpUnLeftQuiv.visible = false;
            this.gpUnRightQuiv.visible = false;
            this.gpArFront.visible = false;
```

```
            this.gpArBack.visible = false;
            this.gpArLeft.visible = false;
            this.gpArRight.visible = false;
            this.gpUnFrontFr.visible = false;
            this.gpUnBackFr.visible = false;
            this.gpUnLeftFr.visible = false;
            this.gpUnRightFr.visible = false;
        }
        if (_gnDirection == "back")
        {
            this.gpArBackAttack.frame = 1;
            this.gpArBackAttack.visible = true;
            this.gpArBackAttack.play();

            // invis everyone else
            this.gpArFrontAttack.visible = false;
            this.gpArRightAttack.visible = false;
            this.gpArLeftAttack.visible = false;
            this.gpUnFrontQuiv.visible = false;
            this.gpUnBackQuiv.visible = false;
            this.gpUnLeftQuiv.visible = false;
            this.gpUnRightQuiv.visible = false;
            this.gpArFront.visible = false;
            this.gpArBack.visible = false;
            this.gpArLeft.visible = false;
```

```
            this.gpArRight.visible = false;
            this.gpUnFrontFr.visible = false;
            this.gpUnBackFr.visible = false;
            this.gpUnLeftFr.visible = false;
            this.gpUnRightFr.visible = false;

    }
    if (_gnDirection == "front")
    {
            this.gpArFrontAttack.frame = 1;
            this.gpArFrontAttack.visible = true;
            this.gpArFrontAttack.play();

            // invis everyone else
            this.gpArBackAttack.visible = false;
            this.gpArRightAttack.visible = false;
            this.gpArLeftAttack.visible = false;
            this.gpUnFrontQuiv.visible = false;
            this.gpUnBackQuiv.visible = false;
            this.gpUnLeftQuiv.visible = false;
            this.gpUnRightQuiv.visible = false;
            this.gpArFront.visible = false;
            this.gpArBack.visible = false;
            this.gpArLeft.visible = false;
            this.gpArRight.visible = false;
```

```
                                        this.gpUnFrontFr.visible = false;
                                        this.gpUnBackFr.visible = false;
                                        this.gpUnLeftFr.visible = false;
                                        this.gpUnRightFr.visible = false;
                                }
                                // play sound
                                if (_soundActive == true)
                                {
                                        swordThrust = new SwordThrust();
                                        swordThrust.play();
                                }
                        }
                }
        }
        else if ((event.keyCode == Keyboard.I)    && (gameMode == "ACTION") )
        {
                dispatchEvent(new Event("gnInvPressed",true));
                gameMode = "INVENTORY";
        }
        else if((event.keyCode == Keyboard.LEFT) && (gameMode == "INVENTORY") )

        {
                dispatchEvent(new Event("gnInvPressedLeft",true));
        }
        else if ((event.keyCode == Keyboard.RIGHT)       && (gameMode ==
"INVENTORY") )
```

```
                {
                        dispatchEvent(new Event("gnInvPressedRight",true));
                }
                else if ((event.keyCode == Keyboard.I)    && (gameMode == "INVENTORY") )
                {
                        dispatchEvent(new Event("gnInvPressed",true));
                        gameMode = "ACTION";
                }
                else if ((event.keyCode == Keyboard.SPACE)       && (gameMode ==
"INVENTORY") )
                {
                        dispatchEvent(new Event("gnInvSelected",true));
                }
        }


        // beigin the "Key press UP" or more specifically...
        // truly it is the "key "RELEASE" function

        private function onKeyUp(event:KeyboardEvent):void
        {
                if (event.keyCode == Keyboard.LEFT || event.keyCode == Keyboard.RIGHT)
                {
                        _vx = 0;
                        //test public variable
                        pubGnVx = _vx;
```

```
		}


		else if (event.keyCode == Keyboard.DOWN || event.keyCode == Keyboard.UP)
		{
			_vy = 0;
			//test public variable
			pubGnVy = _vy;
		}
		else if (event.keyCode == Keyboard.SPACE)
		{
			_shotFired = false;
		}
}



// begin the "ENTER FRAME" events
// as noted in Comment "number 30"
// this entire segment might me regulated by the
// document class controller - note as of march 8th, 2013

public function onEnterFrame(event:Event):void
{
		// begin sword attack animations etc
		if (this.gpArLeftAttack.currentFrame >= 8)
		{
```

```
            this.gpArLeftAttack.visible = false;
            this.gpArLeftAttack.gotoAndStop(1);

            // invis everyone else
            this.gpUnFrontQuiv.visible = false;
            this.gpUnBackQuiv.visible = false;
            this.gpUnLeftQuiv.visible = false;
            this.gpUnRightQuiv.visible = false;
            this.gpArFront.visible = false;
            this.gpArBack.visible = false;
            this.gpArLeft.visible = false;
            this.gpArRight.visible = false;
            this.gpUnFrontFr.visible = false;
            this.gpUnBackFr.visible = false;
            this.gpUnLeftFr.visible = false;
            this.gpUnRightFr.visible = false;
    }
    else if (this.gpArRightAttack.currentFrame >= 8)
    {
            this.gpArRightAttack.visible = false;
            this.gpArRightAttack.gotoAndStop(1);

            // invis everyone else
            this.gpUnFrontQuiv.visible = false;
            this.gpUnBackQuiv.visible = false;
```

```
            this.gpUnLeftQuiv.visible = false;
            this.gpUnRightQuiv.visible = false;
            this.gpArFront.visible = false;
            this.gpArBack.visible = false;
            this.gpArLeft.visible = false;
            this.gpArRight.visible = false;
            this.gpUnFrontFr.visible = false;
            this.gpUnBackFr.visible = false;
            this.gpUnLeftFr.visible = false;
            this.gpUnRightFr.visible = false;
    }
    else if (this.gpArBackAttack.currentFrame >= 8)
    {
            this.gpArBackAttack.visible = false;
            this.gpArBackAttack.gotoAndStop(1);

            this.gpUnFrontQuiv.visible = false;
            this.gpUnBackQuiv.visible = false;
            this.gpUnLeftQuiv.visible = false;
            this.gpUnRightQuiv.visible = false;
            this.gpArFront.visible = false;
            this.gpArBack.visible = false;
            this.gpArLeft.visible = false;
            this.gpArRight.visible = false;
            this.gpUnFrontFr.visible = false;
```

```
            this.gpUnBackFr.visible = false;
            this.gpUnLeftFr.visible = false;
            this.gpUnRightFr.visible = false;
    }
    else if (this.gpArFrontAttack.currentFrame >= 8)
    {
            this.gpArFrontAttack.visible = false;
            this.gpArFrontAttack.gotoAndStop(1);

            this.gpUnFrontQuiv.visible = false;
            this.gpUnBackQuiv.visible = false;
            this.gpUnLeftQuiv.visible = false;
            this.gpUnRightQuiv.visible = false;
            this.gpArFront.visible = false;
            this.gpArBack.visible = false;
            this.gpArLeft.visible = false;
            this.gpArRight.visible = false;
            this.gpUnFrontFr.visible = false;
            this.gpUnBackFr.visible = false;
            this.gpUnLeftFr.visible = false;
            this.gpUnRightFr.visible = false;
    }
```

```
if ((this != null) && (_gnInventory == "NONE") && (this.gpArLeftAttack.visible != true) && (this.gpArRightAttack.visible != true) && (this.gpArBackAttack.visible != true) && (this.gpArFrontAttack.visible != true) )

{

// make sure the quiver is not visible because

// the user has not picked it up yet.

this.gpUnFrontQuiv.visible = false;

this.gpUnBackQuiv.visible = false;

this.gpUnLeftQuiv.visible = false;

this.gpUnRightQuiv.visible = false;

this.gpArFront.visible = false;

this.gpArBack.visible = false;

this.gpArLeft.visible = false;

this.gpArRight.visible = false;

this.gpUnFrontFr.visible = false;

this.gpUnBackFr.visible = false;

this.gpUnLeftFr.visible = false;

this.gpUnRightFr.visible = false;


        // begin the base animated character movment code

        // this should be turned into a function as the inventory

        // for future projects increases.

        // the current long-form of this code is

        // inefficient and hard to read. I know I can do better.

        if(_gnDirection == null)

        {
```

```
            this.gpUnFront.visible = true;

            // the rest are false
            this.gpUnBack.visible = false;
            this.gpUnLeft.visible = false;
            this.gpUnRight.visible = false;

            this.gpUnFront.stop();

}

// move the Gnome

x += _vx;
y += _vy;

//tracings immobilized
//trace(_gnDirection);

if (_gnDirection == "left")
{
            this.gpUnLeft.visible = true;

            // the rest are false
            this.gpUnBack.visible = false;
```

```
            this.gpUnFront.visible = false;
            this.gpUnRight.visible = false;

            if (_vx != 0)
            {
                this.gpUnLeft.play();
            }
            else
            {
                this.gpUnLeft.stop();
            }
        }
        else if (_gnDirection == "right")
        {
            this.gpUnRight.visible = true;

            // the rest are false
            this.gpUnLeft.visible = false;
            this.gpUnBack.visible = false;
            this.gpUnFront.visible = false;


            if (_vx != 0)
            {
                this.gpUnRight.play();
```

```
            }
            else
            {
                this.gpUnRight.stop();
            }
        }
        else if (_gnDirection == "front")
        {
            this.gpUnFront.visible = true;

            // the rest are false
            this.gpUnBack.visible = false;
            this.gpUnLeft.visible = false;
            this.gpUnRight.visible = false;

            if (_vy != 0)
            {
                this.gpUnFront.play();
            }
            else
            {
                this.gpUnFront.stop();
            }
        }
        else if (_gnDirection == "back")
```

```
            {
                        this.gpUnBack.visible = true;

                        // the rest are false
                        this.gpUnFront.visible = false;
                        this.gpUnLeft.visible = false;
                        this.gpUnRight.visible = false;

                        if (_vy != 0)
                        {
                                    this.gpUnBack.play();
                        }
                        else
                        {
                                    this.gpUnBack.stop();
                        }
            }
            // tracings immobilized
            // trace(this);
}
else if ((this != null) && (_gnInventory == "QUIV"))
{
// make sure the quiver is not visible because
// the user has not picked it up yet.
this.gpUnFront.visible = false;
```

```
this.gpUnBack.visible = false;
this.gpUnLeft.visible = false;
this.gpUnRight.visible = false;
this.gpArFront.visible = false;
this.gpArBack.visible = false;
this.gpArLeft.visible = false;
this.gpArRight.visible = false;
this.gpUnFrontFr.visible = false;
this.gpUnBackFr.visible = false;
this.gpUnLeftFr.visible = false;
this.gpUnRightFr.visible = false;


        // begin the base animated character movment code
        // this should be turned into a function as the inventory
        // for future projects increases.
        // the current long-form of this code is
        // inefficient and hard to read. I know I can do better.
        if(_gnDirection == null)
        {
                this.gpUnFrontQuiv.visible = true;

                // the rest are false
                this.gpUnBackQuiv.visible = false;
                this.gpUnLeftQuiv.visible = false;
```

```
            this.gpUnRightQuiv.visible = false;

            this.gpUnFrontQuiv.stop();

}

// move the Gnome

x += _vx;
y += _vy;

//tracings immobilized
//trace(_gnDirection);

if (_gnDirection == "left")
{
            this.gpUnLeftQuiv.visible = true;

            // the rest are false
            this.gpUnBackQuiv.visible = false;
            this.gpUnFrontQuiv.visible = false;
            this.gpUnRightQuiv.visible = false;

            if (_vx != 0)
            {
```

```
                this.gpUnLeftQuiv.play();
            }
            else
            {
                this.gpUnLeftQuiv.stop();
            }
        }
        else if (_gnDirection == "right")
        {
            this.gpUnRightQuiv.visible = true;

            // the rest are false
            this.gpUnLeftQuiv.visible = false;
            this.gpUnBackQuiv.visible = false;
            this.gpUnFrontQuiv.visible = false;



            if (_vx != 0)
            {
                this.gpUnRightQuiv.play();
            }
            else
            {
                this.gpUnRightQuiv.stop();
            }
```

```
}
else if (_gnDirection == "front")
{
        this.gpUnFrontQuiv.visible = true;

        // the rest are false
        this.gpUnBackQuiv.visible = false;
        this.gpUnLeftQuiv.visible = false;
        this.gpUnRightQuiv.visible = false;

        if (_vy != 0)
        {
                this.gpUnFrontQuiv.play();
        }
        else
        {
                this.gpUnFrontQuiv.stop();
        }
}
else if (_gnDirection == "back")
{
        this.gpUnBackQuiv.visible = true;

        // the rest are false
        this.gpUnFrontQuiv.visible = false;
```

```
                    this.gpUnLeftQuiv.visible = false;
                    this.gpUnRightQuiv.visible = false;

                    if (_vy != 0)
                    {
                            this.gpUnBackQuiv.play();
                    }
                    else
                    {
                            this.gpUnBackQuiv.stop();
                    }
            }
}
// BEGIN THE CODE FOR "SPELL"
else if ((this != null) && (_gnInventory == "SPELL"))
{
// make sure the quiver is not visible because
// the user has not picked it up yet.
this.gpUnFront.visible = false;
this.gpUnBack.visible = false;
this.gpUnLeft.visible = false;
this.gpUnRight.visible = false;
this.gpArFront.visible = false;
this.gpArBack.visible = false;
this.gpArLeft.visible = false;
```

```
this.gpArRight.visible = false;
this.gpUnFrontQuiv.visible = false;
this.gpUnBackQuiv.visible = false;
this.gpUnLeftQuiv.visible = false;
this.gpUnRightQuiv.visible = false;

        // begin the base animated character movment code
        // this should be turned into a function as the inventory
        // for future projects increases.
        // the current long-form of this code is
        // inefficient and hard to read. I know I can do better.
        if(_gnDirection == null)
        {
                this.gpUnFrontFr.visible = true;

                // the rest are false
                this.gpUnBackFr.visible = false;
                this.gpUnLeftFr.visible = false;
                this.gpUnRightFr.visible = false;

                this.gpUnFrontFr.stop();

        }

        // move the Gnome
```

```
x += _vx;
y += _vy;

//tracings immobilized
//trace(_gnDirection);

if (_gnDirection == "left")
{
	this.gpUnLeftFr.visible = true;

	// the rest are false
	this.gpUnBackFr.visible = false;
	this.gpUnFrontFr.visible = false;
	this.gpUnRightFr.visible = false;

	if (_vx != 0)
	{
		this.gpUnLeftFr.play();
	}
	else
	{
		this.gpUnLeftFr.stop();
	}
}
```

```
else if (_gnDirection == "right")
{
        this.gpUnRightFr.visible = true;

        // the rest are false
        this.gpUnLeftFr.visible = false;
        this.gpUnBackFr.visible = false;
        this.gpUnFrontFr.visible = false;



        if (_vx != 0)
        {
                this.gpUnRightFr.play();
        }
        else
        {
                this.gpUnRightFr.stop();
        }
}
else if (_gnDirection == "front")
{
        this.gpUnFrontFr.visible = true;

        // the rest are false
        this.gpUnBackFr.visible = false;
```

```
		this.gpUnLeftFr.visible = false;
		this.gpUnRightFr.visible = false;

		if (_vy != 0)
		{
			this.gpUnFrontFr.play();
		}
		else
		{
			this.gpUnFrontFr.stop();
		}
	}
	else if (_gnDirection == "back")
	{
		this.gpUnBackFr.visible = true;

		// the rest are false
		this.gpUnFrontFr.visible = false;
		this.gpUnLeftFr.visible = false;
		this.gpUnRightFr.visible = false;

		if (_vy != 0)
		{
			this.gpUnBackFr.play();
		}
```

```
                    else
                    {
                        this.gpUnBackFr.stop();
                    }
                }
            }
                /// begin the "SWORD" inventory mode

                else if ((this != null) && (_gnInventory == "SWORD") &&
(this.gpArLeftAttack.visible != true) && (this.gpArRightAttack.visible != true) &&
(this.gpArBackAttack.visible != true) && (this.gpArFrontAttack.visible != true))
                {
                // make sure the quiver is not visible because
                // the user has not picked it up yet.
                this.gpUnFront.visible = false;
                this.gpUnBack.visible = false;
                this.gpUnLeft.visible = false;
                this.gpUnRight.visible = false;
                this.gpUnFrontQuiv.visible = false;
                this.gpUnBackQuiv.visible = false;
                this.gpUnLeftQuiv.visible = false;
                this.gpUnRightQuiv.visible = false;
                this.gpUnFrontFr.visible = false;
                this.gpUnBackFr.visible = false;
                this.gpUnLeftFr.visible = false;
                this.gpUnRightFr.visible = false;
```

```
// begin the base animated character movment code
// this should be turned into a function as the inventory
// for future projects increases.
// the current long-form of this code is
// inefficient and hard to read. I know I can do better.
if(_gnDirection == null)
{
        this.gpArFront.visible = true;

        // the rest are false
        this.gpArBack.visible = false;
        this.gpArLeft.visible = false;
        this.gpArRight.visible = false;

        this.gpArFront.stop();

}

// move the Gnome

x += _vx;
y += _vy;

//tracings immobilized
```

```
//trace(_gnDirection);


if (_gnDirection == "left")
{
	this.gpArLeft.visible = true;

	// the rest are false
	this.gpArBack.visible = false;
	this.gpArFront.visible = false;
	this.gpArRight.visible = false;

	if (_vx != 0)
	{
		this.gpArLeft.play();
	}
	else
	{
		this.gpArLeft.stop();
	}
}
else if (_gnDirection == "right")
{
	this.gpArRight.visible = true;

	// the rest are false
```

```
            this.gpArLeft.visible = false;
            this.gpArBack.visible = false;
            this.gpArFront.visible = false;


            if (_vx != 0)
            {
                  this.gpArRight.play();
            }
            else
            {
                  this.gpArRight.stop();
            }
}
else if (_gnDirection == "front")
{
            this.gpArFront.visible = true;

            // the rest are false
            this.gpArBack.visible = false;
            this.gpArLeft.visible = false;
            this.gpArRight.visible = false;

            if (_vy != 0)
            {
```

```
                this.gpArFront.play();
            }
            else
            {
                this.gpArFront.stop();
            }
        }
        else if (_gnDirection == "back")
        {
            this.gpArBack.visible = true;

            // the rest are false
            this.gpArFront.visible = false;
            this.gpArLeft.visible = false;
            this.gpArRight.visible = false;

            if (_vy != 0)
            {
                this.gpArBack.play();
            }
            else
            {
                this.gpArBack.stop();
            }
        }
```

```
                    // tracings immobilized
                    // trace(this);
            }
    }
    private function shootProjectile():void
    {
            var projRotation:Number;
            var projectile_Vx:Number;
            var projectile_Vy:Number;
            // projectile velocity
            // and projectile direction
            if (_gnDirection == "front")
            {
                    projRotation = 90;
                    projectile_Vx = 0;
                    projectile_Vy = 15;
            }
            else if (_gnDirection == "back")
            {
                    projRotation = 270;
                    projectile_Vx = 0;
                    projectile_Vy = -15;
            }
            else if (_gnDirection == "right")
            {
```

```
                projRotation = 0;
                projectile_Vx = 15;
                projectile_Vy = 0;
            }
            else if (_gnDirection == "left")
            {
                projRotation = 180;
                projectile_Vx = -15;
                projectile_Vy = 0;
            }


            // prjectile start position

            var projectile_StartX:Number = x;
            var projectile_StartY:Number = y;

            // create projectile instance and add it to the stage

            var projectile:Projectile = new Projectile(projectile_Vx, projectile_Vy,
projectile_StartX, projectile_StartY, projRotation, _gnInventory);
            parent.addChild(projectile);

        }
    }
```

```
}
```

SECTION 3 – PROJECTILE CLASS

```
package
{
	import flash.display.MovieClip;
	import flash.events.Event;

	public class Projectile extends MovieClip
	{
		// delinate ammo types here

		private const ARROW:uint = 1;
		private const SPELL:uint = 2;

		// set up variables

		private var _vx:Number;
		private var _vy:Number;
		private var _projX:Number;
		private var _projY:Number;
		private var _projectileType:String;
```

```
		private function onGnDirection(event:Event)
		{
			trace("this is coming from the projectile class");



		}
		public function Projectile(vx:Number, vy:Number, projX: Number, projY:Number , projRot:Number, projectileType:String)
		{
			// test the function
			this._vx = vx;
			this._vy = vy;

			// assign velocity and start postion
			this.x = projX;
			this.y = projY;

			this.rotation = projRot;

			// adjust the projectile type

			switch (projectileType)
			{
				case "QUIV" :
					gotoAndStop(1);
```

```
                        break;


                case "SPELL" :
                        gotoAndStop(2);
                        this._vx = this._vx * 0.8;
                        this._vy = this._vy * 0.8;
                        break;
        }
        // set the projectiletype

        this._projectileType = projectileType;




        // add an event listener called: "Added to stage etc"
        addEventListener(Event.ADDED_TO_STAGE, onAddedToStage);

        // add event listener for the GpUnarmed location

}
private function onAddedToStage(event:Event):void
{
        // add your listeners
        addEventListener(Event.ENTER_FRAME, onEnterFrame);
        addEventListener(Event.REMOVED_FROM_STAGE, onRemovedFromStage);
```

```
            // Tell gnome prince that his projectile has been created
            dispatchEvent(new Event("projectileCreated", true));

            // remove onAddedToStage Listener
            removeEventListener(Event.ADDED_TO_STAGE, onAddedToStage);
    }
    private function onRemovedFromStage(event:Event):void
    {
            removeEventListener(Event.ENTER_FRAME, onEnterFrame);
            removeEventListener(Event.REMOVED_FROM_STAGE, onRemovedFromStage);
    }
    private function onEnterFrame(event:Event):void
    {
            // consider adding projectile rotation here

            // move the projectile

            x += _vx;
            y += _vy;
    }

    // getters

    public function get projectileType():String
```

```
		{
			return _projectileType;
		}
		public function get projectileVx():int
		{
			return _vx;
		}
		public function get projectileVy():int
		{
			return _vy;
		}
	}

}
```