Comp 388/488 - Game Design and Development

Spring Semester 2019 - Week 9

Dr Nick Hayward

Games and formal elements

importance of objectives

- objectives may help establish different requirements and goals in a game
  - *helping a user to achieve results within the confines of the rules of the game*
- objectives may seem challenging and difficult
  - *need to be correctly designed relative to a game's rules*
  - *they should also seem achievable to a player*
- a game's objective may also help set the tone for gameplay and interaction
  - *e.g. objective of most platform games different from sports-based game*
  - *tone for each of these games becomes a reflection of the objective*
- use of objectives in games is not limited to just the overall game itself
- may consider defining an objective for different player roles
  - *or perhaps as mini challenges within our games*
- each level may define its own objective
  - *such as completing a level as fast as possible*
  - *collecting all available options (coins, for example) on another*
- choice of such objectives needs to be considered carefully
- each will affect not only the formal system of our game
  - *but also the dramatic aspect*
- good integration of objectives in the premise or story of a game
  - *helps strengthen dramatic aspects*
  - *e.g. Legend of Zelda*

Games and formal elements

a consideration of procedures

- we may start to see a few common actions that exist across multiple genres
- these often include the following:
  - *an action to start the game*
    - specific procedure required to initiate gameplay...
  - *ongoing actions and procedures*
    - e.g. common, persistent actions that continue, repeat &c. as part of the game
  - *reserved or special actions*
    - e.g. actions that may be required and executed due to a given condition or game requirement
  - *actions to conclude or resolve*
    - e.g. resolve actions at certain points within the game, or at the end of the game itself...
- for video games, incl. consoles, mobile, PC...
  - *such actions and procedures closely associated player interactions*
  - *e.g. given key combinations or controller buttons*
  - *perhaps tapping particular options on the screen itself*
  - *or moving a mobile device to control certain actions...*
- consider Super Mario Bros.
  - *we may clearly identify controls for given actions and procedures*
    - expected usage for directional buttons
    - option to jump or swim with the **A** button, &c.

Games and formal elements

procedures in development

- procedures also play a key role in the way we develop our games
- we can add procedures within the logic of our game
  - *to monitor certain ongoing states, user interaction, updates, rendering...*
- these procedures are working in the core of our game
  - *responding to changes in state*
- e.g. a player completes a puzzle within the main game
  - *need to monitor the ongoing puzzles responses*
  - *check the player input and interactions*
  - *then update the state of the game in response to a success or failure result*
- we're effectively checking whether a given action succeeds or not
  - *then, determine impact this may have on the game itself*
- such procedures and actions are naturally limited by real-world constraints
  - *e.g. performance of the underlying system, controllers, interaction options, screen...*
- may need to tailor such requirements to match the type of game we're developing
  - *and the target audience...*

Python and Pygame - Game Example 1

animating sprite images - part 1

- for many game sprites it's fun and useful to add different animations
  - *to animate different states, actions, &c. as the game progresses...*
  - *e.g. random rotation of mob objects, explosions, collisions...*
- already added scale transform to mob objects
  - *we may use the same pattern to add a rotate option*
  - *add animation to these sprites as they move down the game window*
  - *e.g. start by setting some variables for our rotation,*

```
# set up rotation for sprite image - default rotate value, rotate speed to add diff. direc
self.rotate = 0
self.rotate_speed = random.randrange(-7, 7)
```

- due to the framerate of this game, set to 60FPS
  - *need to ensure rotate animation does not occur for each update of the game loop*
  - *if not, rotation will be too quick, unrealistic, annoying...*

Python and Pygame - Game Example 1

animating sprite images - part 2

- in addition to the rotate animation
  - *also need to consider how to create a timer for this animation*
  - *regularity of update to the animation to ensure it renders realistically*
- already a timer available within our existing code
  - *currently using to monitor the framerate for our game*
- use this timer to check the last time we updated our mob sprite image
- set a time to rotate the sprite image
  - *then check this monitor as it reaches this specified time*
- record last time our sprite image was rotated by getting the time
  - *number of ticks since the game started, e.g.*

```
# check timer for last update to rotate
self.rotate_update = pygame.time.get_ticks()
```

- each time the mob sprite image object is rotated
  - *update value of variable to record the last time for a rotation*
  - *modify the mob sprite's `update` function as follows,*

```
# call rotate update
self.rotate()
```

- simply going to call a separate rotate function
  - *keep the code cleaner and easier to read*
  - *allows us to quickly and easily modify, remove, and simply stop our object's rotation*

rotate

- now add our new `rotate()` function
  - *start by checking if it's time to rotate the sprite image*

```
def rotate(self):
      # check time - get time now and check if ready to rotate sprite image
      time_now = pygame.time.get_ticks()
      # check if ready to update...in milliseconds
      if time_now - self.rotate_update > 70:
          self.last_update = time_now
```

- uses the current time, relative to the game's timer
  - *then checks this value against the last value for a rotate update*
- if difference is greater than 70 milliseconds
  - *it's time to rotate the sprite object*

rotate issues

- for rotation we can't simply add a *rotate transform* to the `rotate()` function
  - *possible in the code, it will also cause the game window to practically freeze*
  - *makes the game unplayable in most examples, e.g.*

```
self.image = pygame.transform.rotate(self.image, self.rotate_speed)
```

- this issue is due to pixel loss for the image
- each rotation of a sprite object image
  - *causes a game's logic to lose part of the pixels for that image*
- this will cause the game loop to start to freeze...

correct rotation

- correct this rotation issue by working with an original, pristine image for the sprite object

```
# set pristine original image for sprite object
self.image_original = mob_img
# set colour key for original image
self.image_original.set_colorkey(BLACK)
```

- then, set the initial sprite object image as a copy of this original

```
# set copy image for sprite rendering
self.image = self.image_original.copy()
```

- then, we may use the pristine original image with the rotation

```
self.image = pygame.transform.rotate(self.image_original, self.rotate_speed)
```

Python and Pygame - Game Example I

correct rotation speed

- another issue we need to fix is the rotation speed for a sprite object image
- if we simply use our default `self.rotate_speed`
  - *not keeping track of how far we've actually rotated the image*
- need to keep a record of incremental rotation of the image
  - *ensure that it rotates smoothly and in a realistic manner*
- monitor this rotation by using the value of the rotation
  - *adding rotation speed for each update to a sprite object image*
- as the image rotates we can simply check its value as a modulus of 360
  - *to ensure it keeps rotating correctly*

```
self.rotate = (self.rotate + self.rotate_speed) % 360
self.image = pygame.transform.rotate(self.image_original, self.rotate)
```

Python and Pygame - Game Example 1

rect rotation issues

- still have an issue with the rectangle bounding box, which does not rotate
- as sprite image rotates, it loses its centre relative to the bounding rectangle
- to correct this issue, we can now modify our logic for the sprite object's *update*, e.g.

```
# new image for rotation
rotate_image = pygame.transform.rotate(self.image_original, self.rotate)
# check location of original centre of rect
original_centre = self.rect.center
# set image to rotate image
self.image = rotate_image
# create new rect for image
self.rect = self.image.get_rect()
self.rect.center = original_centre
```

- mob sprite object images will now correctly rotate as they move down the screen

resources

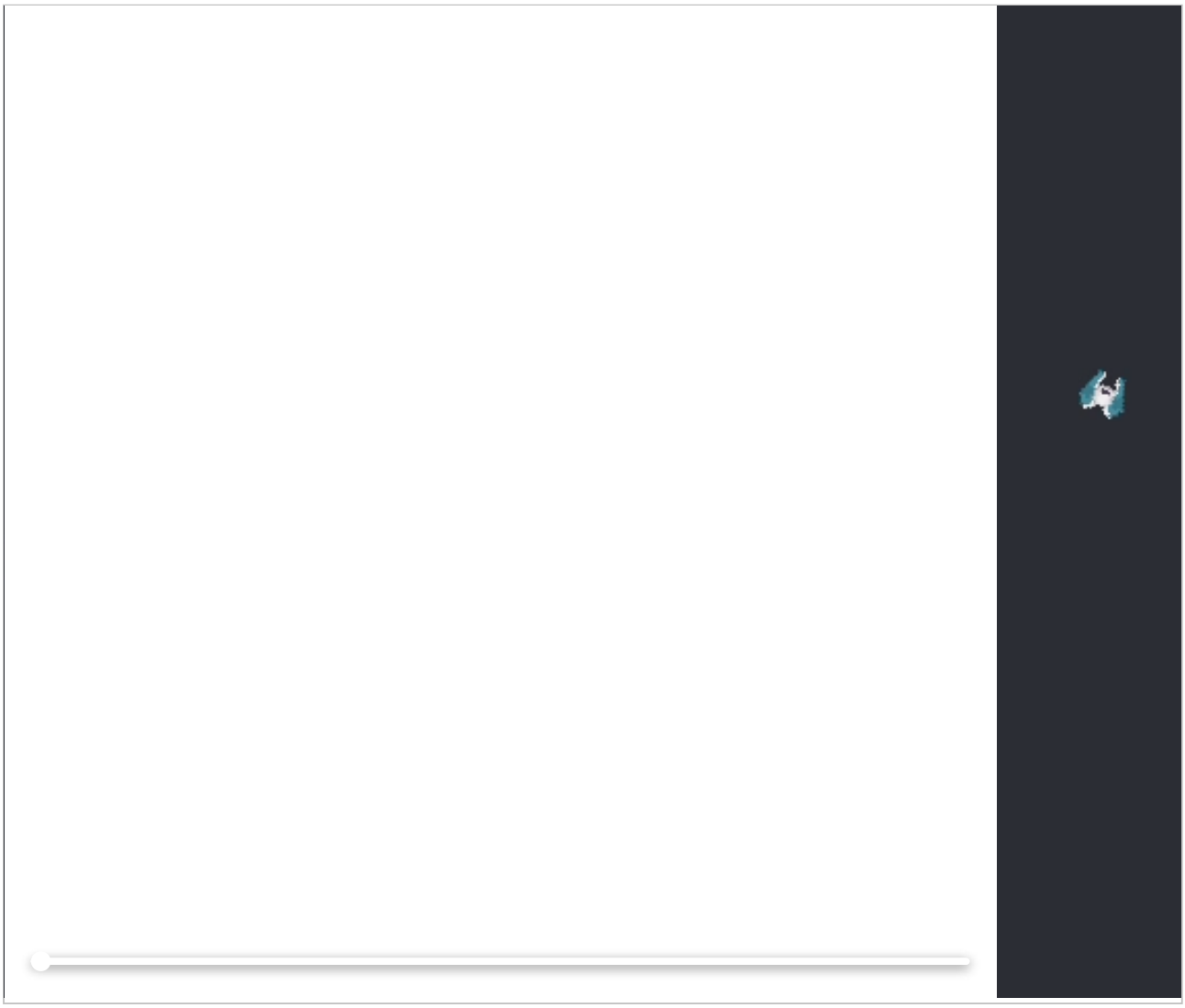
- notes = sprites-animating-images.pdf
- code = animatingsprites1.py

game example

- shooter0.6.py
  - *animating sprite images*
    - rotate mob images down the screen
    - create pristine image for rotation
    - update rect bounding box to ensure it rotates correctly

Video - animating sprites

rotation



Video - Shooter 0.6

animating sprite images

Games and formal elements

rules and game concepts

- as we define and formalise rules for our games
  - *need to consider more than simply the gameplay itself*
- objects in games, and concepts embedded in gameplay structures
  - *require defined limitations and rules*
- game objects
  - *characters, weapons, vehicles, obstacles...*
  - *may be derived or inspired by real world objects*
- objects may come with the perception of existing limitations and rules
  - *a player knows what these objects can and cannot do in the real world*
- we may use these real world objects as inspiration
  - *starting points for our game's objects*
  - *not inherently limited or defined by them*
  - *may modify as befits the requirements of our game, and its gameplay*
- game context will be a determining factor in development of our objects
- objects may also be developed as a group of properties and variables
  - *together form the whole from varying requirements*
- in a world of chivalry, knights, ogres, and other fantastical creatures
  - *we may still create concepts and objects that unify these characters*
  - *from base objects, we can simply inherit and modify as needed*
- e.g. we may require various characters to ride
  - *on horseback, or perhaps astride an elephant, or even a fictional dragon &c.*
- our objects may be abstracted to include known attributes
  - *which can then be used as the parent*
  - *use for multiple real and imagined objects, characters within our game*

Games and formal elements

rules, objects, and updates...

- as developers and designers
  - *need to ensure a balance between maintaining game objects and variables*
  - *creating an intuitive update for our users*
- unlikely our player will want to keep a manual tally of such updates
  - *need to consider how we may allow them to quickly and easily intuit game objects*
- for example, we may need to
  - *maintain a running total of game objects, such as coins, lives, energy levels*
  - *correctly inform the player of any updates*
- a player should be able to quickly learn the nature of these objects
- if they're too difficult or complex
  - *need to consider how this affects our player's gaming experience*
- also need to ensure that there is sufficient isolation between different objects
- a player should be able to discern differences without too much effort or guesswork
- updates may also be influenced by known restrictions in the game's rules
  - *useful in many respects*
  - *e.g. relative to boundaries, objectives, and objects themselves*
- by establishing rules, e.g.
  - *to restrict objects and their attributes*
- rules help create a known scale for state within our game
- player has defined restrictions
  - *they know what they can and can't do*
  - *risk and reward is set in the game's logic and gameplay*

Python and Pygame - Game Example 1

random mob sprite images

- as we add sprite image objects to a game window, e.g. multiple *mob* images
  - *we can make the game experience more fun*
  - *by randomising the image for each mob sprite object*
- we may use a group of images as possible mob images
  - *then randomise their selection for each new mob sprite object image*
- to add random image, at least randomised from potential options...
  - *need to add a list of available images for the random selection, e.g.*

```
asteroid_list = ["asteroid-tiny-grey.png", "asteroid-small-grey.png", "asteroid-med-grey.p
```

- also need a new list for our asteroid images, e.g.

```
asteroid_imgs = []
```

- simply need to loop through this asteroid list
  - *then add each available image to the list of `asteroid_imgs`, e.g.*

```
for img in asteroid_list:
    asteroid_imgs.append(pygame.image.load(os.path.join(img_dir, img)).convert())
```

- then update the `Mob` class to set a random image from `asteroid_imgs` list, e.g.

```
self.image_original = random.choice(asteroid_imgs)
```

- images for our mob sprite objects will now be randomly chosen from the available list of images

resources

- notes = sprites-animating-random-images.pdf
- code = animatingsprites2.py

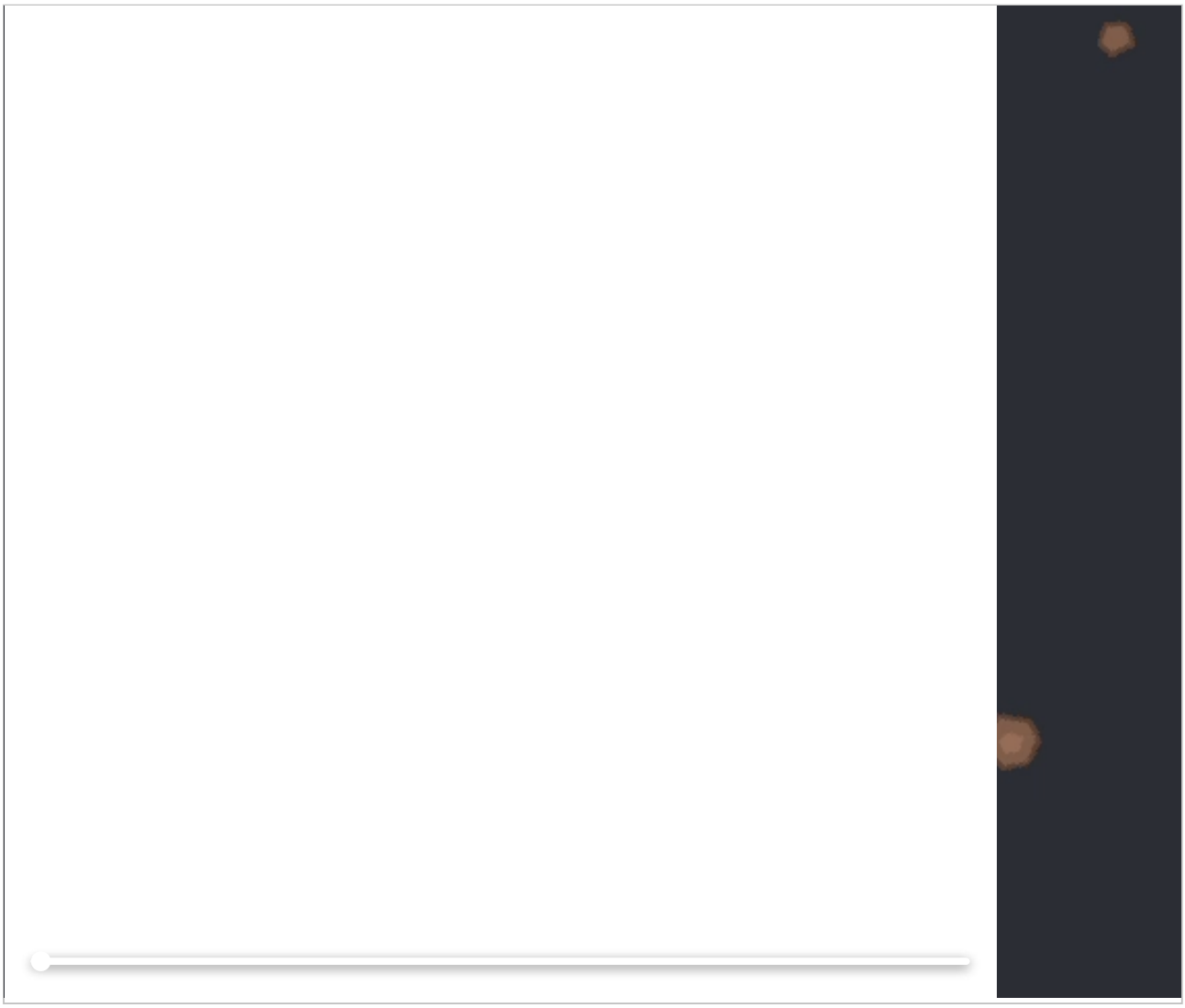
game example

- shooter0.7.py
- set random image for mob sprite object image
  - *random image from selection of image options*

- *rotate and animate each random mob sprite image*

Video - Animating Sprites

random mob images



Video - Shooter 0.7

set random image for mob sprite object image

Game designers

Designer example - Will Wright

- Wright is a veteran American game designer
  - *best known for his work on The Sims*
- *The Sims* was originally released in 2000
  - *led to countless versions, spin-offs &c.*
  - *driven a genre more interested in participation than a definitive win*
- as a co-founder of *Maxis*, and then later part of EA
  - *Wright also developed the game Spore*
- he's often referred to as a designer of *software toys* instead of traditional games
  - *a consideration of the non-traditional structure employed for many of his games*
- he's also been a passionate developer of, and advocate for, emergent and adaptive systems
- Wright has continued to develop this concept for many of his games
  - *his legacy is evident in games such as Spore, The Sims 3 and The Sims 4*
- Wright has tried to use these systems with their simple rules and definitions
  - *to provide the possibility for the development of complex, detailed outcomes*

Resources

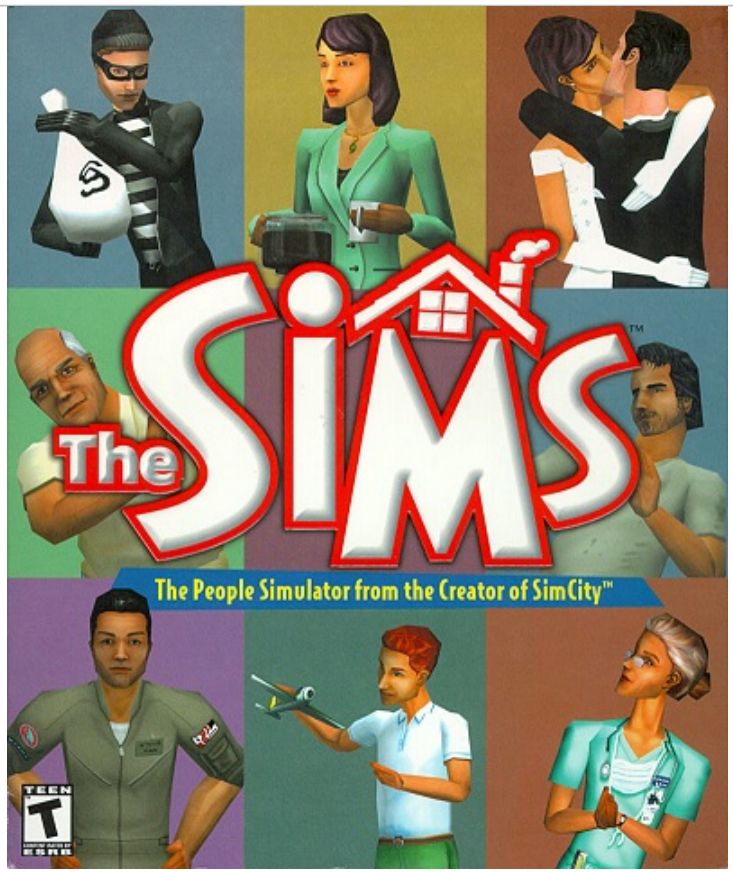
- Maxis
- The Sims
- Spore
- Will Wright

Image - Will Wright

Will Wright

Image - Will Wright

The Sims and Spore



The Sims

Video - Will Wright

The Sims Creator...

- Summit on Science, Entertainment, and Education - Will Wright - Vimeo

Python and Pygame - Game Example I

render text to a game window - intro

- drawing text to a game window in Pygame can become a repetitive process
  - *in particular, as part of each window update*
- we may abstract this underlying game requirement to a text output function

```
 # text output and render function - draw to game window
def textRender(surface, text, size, x, y):
    # specify font for text render
    ...
```

- start by specifying a surface where we need to draw the text
  - *plus text to render, its size, and coordinates relative to surface*
- need to specify a font for the text to be rendered
  - *reliant upon installed fonts for user's local system*
- use a font-match function with Pygame
  - *helps abstract specification of exact font to a relative name*

```
# specify font name to find
font_match = pygame.font.match_font('arial')
```

- Pygame will search local system for a font with the specified name
- use specified font to create an object for the font
  - *we need this object to render text in the game window*

```
# specify font for text render - uses found font and size of text
font = pygame.font.Font(font_match, size)
```

Python and Pygame - Game Example I

render text to a game window - text drawing

- text we'll be adding to the game window needs to be drawn
  - *drawn effectively pixel by pixel*
- Pygame calculates drawing for each pixel
  - *creates the specified text in the required font*
- start by specifying a surface to draw the required pixels for the text, e.g.

```
# surface for text pixels - TRUE = anti-aliased
text_surface = font.render(text, True, WHITE)
```

- we're specifying where to draw the text
  - *the text to draw to the game window*
  - *a boolean value for anti-aliasing of text*
  - *and the text colour*
- need to calculate a rectangle for placing the text surface, e.g.

```
# get rect for text surface rendering
text_rect = text_surface.get_rect()
```

- then specify where to position our text surface
  - *relative to defined `x` and `y` coordinates, e.g.*

```
# specify a relative location for text
text_rect.midtop = (x, y)
```

- text is then added to the surface using the standard `blit` function, e.g.

```
# add text surface to location of text rect
surface.blit(text_surface, text_rect)
```

Python and Pygame - Game Example 1

render text to a game window - text draw function

- overall text draw function is now as follows,

```
# text output and render function - draw to game window
def textRender(surface, text, size, x, y):
    # specify font for text render - uses found font and size of text
    font = pygame.font.Font(font_match, size)
    # surface for text pixels - TRUE = anti-aliased
    text_surface = font.render(text, True, WHITE)
    # get rect for text surface rendering
    text_rect = text_surface.get_rect()
    # specify a relative location for text
    text_rect.midtop = (x, y)
    # add text surface to location of text rect
    surface.blit(text_surface, text_rect)
```

- call this function whenever we need to render text to our game window
- in `draw` section of our game loop, now add the following call, e.g.

```
# draw text to game window - game score
textRender(window, str(game_score), 16, winWidth / 2, 10)
```

Python and Pygame - Game Example 1

render text to a game window - add a game score

- common example of rendering text in a game window
  - *simply output a running score for the player*
- start by adding an initial variable to record the player's score, e.g.

```
# initialise game score - default to 0
game_score = 0
```

- then allow a player to score points for each projectile collision on a mob object
  - *e.g. laser beam hit on an asteroid*
  - *fun to set variant points relative to size of mob object*
- if we use the radius of each mob object
  - *perform a quick calculation for each* `collision`
  - *work out points per asteroid, e.g.*

```
# calculate points relative to size of mob object
game_score += 40 - collision.radius
```

- relative to the recorded `collision`
  - *simply get the radius per hit mob object*
  - *then minus from a known starting value*

resources

- notes = drawing-text.pdf
- code
- drawingtext1.py (game example with score)
- drawingtext2.py (abstracted - simple rendered text)

game example

- shooter0.8.py
- draw text to the game window
- keep a running score for collisions with a projectile
  - *player shoots and destroys an asteroid*

- *score is calculated relative to size of mob object - radius...*

- score is rendered to top of game window
  - *update for each successful hit*

Video - Shooter 0.8

render text for a game score

Games and dramatic elements

intro

- may consider *dramatic elements* as we continue to design and develop our games
- already considered many underlying elements and concepts that create a game we recognise
- also need to consider those elements that create...
  - *a sense of emotion,*
  - *engagement*
  - *and challenge for our players*
- aspects of our game that encourage an emotional connection
  - *simple desire to invest time and effort in gameplay*
- **dramatic elements** help create a sense of context to a player's experience with our game
- **dramatic elements** provide a backdrop/overlay for our game
  - *combines many disparate formal elements of our game logic and development*
  - *creates a conceptually meaningful experience for the player*
- may start with universal concepts for such dramatic elements
  - *including challenge and play*
- then branch out into more complicated considerations of elements, e.g.
  - *characters, premise, story...*
  - *used by most games we design, develop, and play*
- used to form core for explaining many of more abstract elements of a game's formal system
- help create a deeper sense of connection between the game and its player

Games and dramatic elements

gaming challenge

- *challenge* and an associated sense of accomplishment
  - *fundamental definition of gaming for many players*
  - *perception of worthwhile gaming experience*
- challenge alone is often no different from work, daily issues...
- designers need to find a happy balance to challenge and reward
- need to consider tasks that are satisfying to complete and provide a balance between work and fun
- designers are inherently limited by the abilities and skills of an individual player
- challenge may also become an individual perception and characteristic of a player
  - *consider difference between age groups, skill levels, experience...*
- challenge may also be considered *dynamic*
  - *a player's ability will adapt and improve*
  - *hopefully as they learn and progress through a game*
- a challenging early task may become considerably easier
  - *i.e. as a player progresses to subsequent levels and areas within a game*
- as a player learns these new skills
  - *enjoys opportunity to test and demonstrate these skills elsewhere in the game*
- incremental modifications and updates to earlier, completed challenges
  - *provides a quick and easy option for the player to balance challenge with reward*
- designers and developers need to consider challenge carefully
  - *challenge that is not necessarily defined by individual experience*

Games and dramatic elements

a sense of flow

- carefully consider how to design our games to effectively consider *challenge*
  - *as defined and restricted by individual experience, &c.*
- each experience can, therefore, take advantage of an appropriate level of challenge
- a well-known example of this was developed by the psychologist **Mihaly Csikszentmihalyi**
- he wanted to identify concepts and elements that might help define enjoyment for a given task
  - *he studied experiences and similarities of various tasks for different people*
  - *trying to discern similarities of experience for these tasks, players...*
- his research noted a distinct lack of traditionally perceived bias
  - *for what we consider fun and meaningful tasks*
  - *lack of bias in results for age, social standing, gender...*
- people simply described their perception of enjoyable activities in a similar manner
- regardless of the activity itself
  - *often included disparate pursuits such as music, painting, and playing games...*
  - *the words and concepts people used to articulate this sense of fun was largely the same*
- for each of these tasks
  - *certain conditions became recurrent and popular for describing pleasurable activities*
  - *each user and player was entering into a state of **flow***
  - *allowed for this heightened sense of achievement, and associated fun*

perceptions of flow

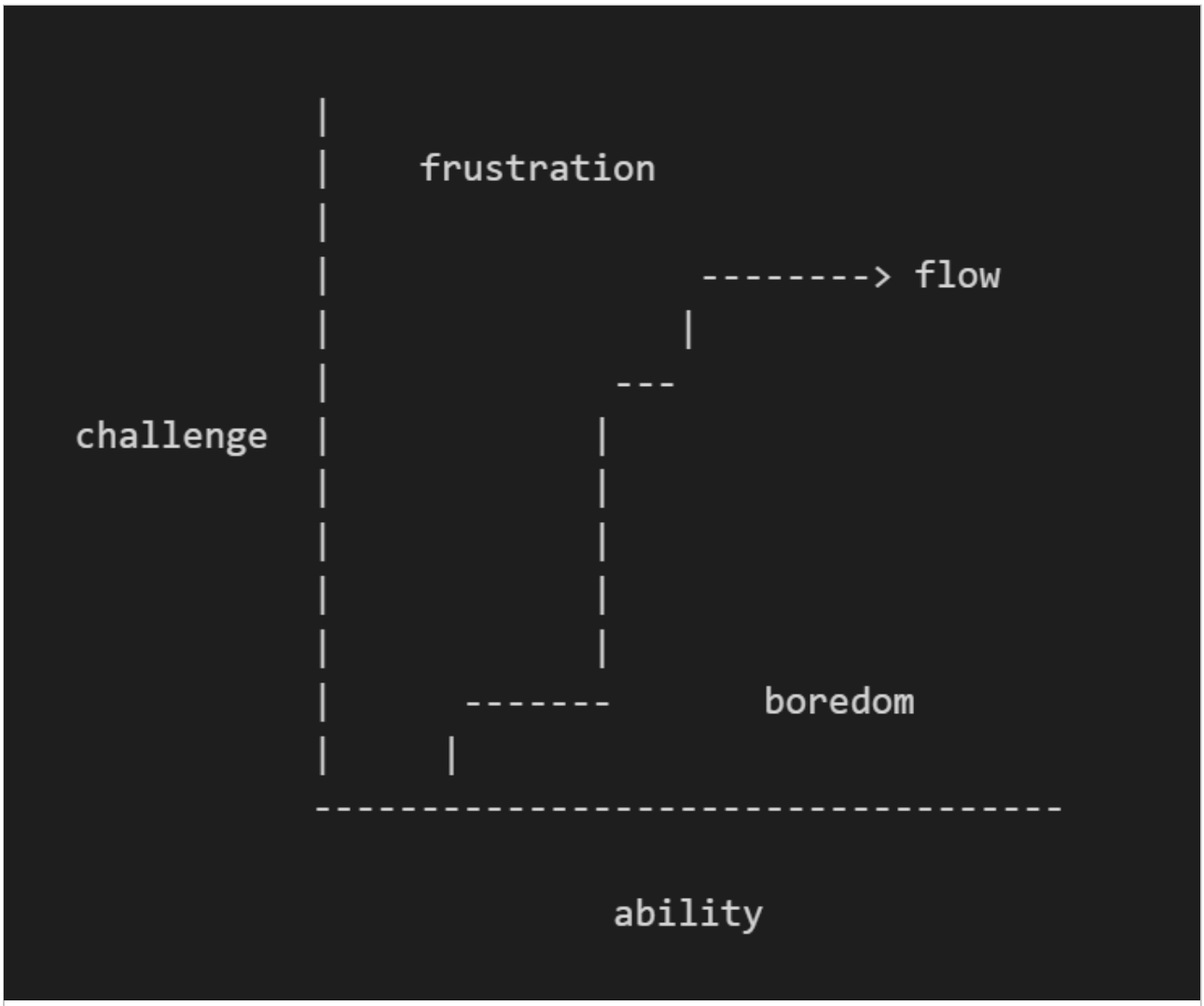
Flow by **Mihaly Csikszentmihalyi**

- player's creativity, ability, and general awareness are high
  - *performance of activity occurs naturally and unconsciously*
- player experiences deep concentration and immersion in their current activity
  - *player is effectively both alert and relatively relaxed*
- living in the moment
  - *a sensation of being so engrossed in an activity a player is unaware of the passage of time*
- balancing interest and challenge
- player is confident and exhibits a sense of control over their current situation
- player is working progressively towards achieving a specific goal, e.g.
  - *getting to the next level in a game*
  - *completing a mini-challenge*
  - *or mastering a particular mechanic for their current character*
  - *Luigi's Mansion and the vacuum cleaner...*

TED 2004 - Flow, the secret to happiness

Image - Games and dramatic elements

a state of flow



A state of flow

Video - Colin McRae Rally

Source - Colin McRae Rally, YouTube

Fun and Games

Driving game example

- Colin McRae Rally - Playstation

Python and Pygame - Game Example I

game music and sound effects - intro

- most of these sound effects will use a *WAV* format
  - *may also use other file formats such as OGG*
- add these files for our sound effects to the game assets directory, e.g.

```
|-- shootemup
    |-- assets
        |-- images
            |__ ship.png
        |-- sounds
            |__ laser-beam-med.wav
            |__ explosion-med.wav
```

Python and Pygame - Game Example I

game music and sound effects - import sounds and effects

- we need to add support for Pygame's `mixer`
  - *add the following call after we initialise Pygame itself, e.g.*

```
# add sound mixer to game
pygame.mixer.init()
```

- to use these sounds and effects in our game window
  - *need to add the directory location, e.g.*

```
# relative path to music and sound effects dir
snd_dir = os.path.join(assets_dir, "sounds")
```

- then start to add our required music and sound effects, e.g.

```
# load music and sound effects for use in game window
# laser beam firing sound effect
laser_effect = pygame.mixer.Sound(os.path.join(snd_dir, 'laser-beam-med.wav'))
# explosion sound effect
explosion_effect = pygame.mixer.Sound(os.path.join(snd_dir, 'explosion-med.wav'))
```

- add these lines of code right after we've loaded our images
  - *just before we start the game loop itself*

Python and Pygame - Game Example I

game music and sound effects - use sound effects

- after importing and loading our sound effects
  - *we may then choose where we need to play these sound effects in our game*
  - *e.g. player fires a laser beam to destroy falling mob objects*

```
# fire projectile from top of player sprite object
def fire(self):
    ...
    # play laser beam sound effect
    laser_effect.play()
```

- also add sound effects for each mob object explosion

```
# play laser beam sound effect
laser_effect.play()
```

Python and Pygame - Game Example 1

game music and sound effects - use music in a game

- as we add sound effects, we may also load music to play in the game
  - *we may add background music for the game window, e.g.*

```
# load music for background playback in game window
pygame.mixer.music.load(os.path.join(snd_dir, 'space-music-bg.ogg'))
```

- also set a relative volume for this background music
  - *creates ambience and does not overwhelm the player experience, e.g.*

```
# set music volume - half standard volume
pygame.mixer.music.set_volume(0.5)
```

resources

- notes = music-intro.pdf
- code
- basicmusic1.py
- basicmusic2.py

game example

- shooter0.9.py
- add music and sound effects to the game window
  - *add pygame mixer*
  - *load sounds directory in assets*
  - *load required sounds and sound effects*
  - *call `play()` for each required sound effect and game music...*

Video - Shooter 0.9

add music and sound effects

Games and dramatic elements

consider skills

- start introducing challenges and associated activities into our games that require definable skills
- may be a mixture of assumed or learnt skills, applicable to the current game
- for *flow*, **Csikszentmihalyi** describes it relative to activities that are considered,

> *goal-directed and bounded by rules...*

- Csikszentmihalyi, M. *Flow: The Psychology of Optimal Experience*. Harper & Row. New York. 1990. P.49.
- such activities not customarily achieved or completed without proper requisite skills
- skills may include various examples, including
  - *standard motor skills for controls and interaction*
  - *problem solving*
  - *social interaction with other players...*
- challenges, and the development of skills, need not necessarily be limited
  - *e.g. by simple clicking of buttons, and the resultant moving of pixels...*
- a common trick to manipulate such skills is the introduction of doubt or variance
- imagine a challenge or task where the ending is not known or guaranteed
  - *e.g. a player's character walking along a ledge*
  - *may be wet underfoot*
  - *perception of wind blowing from any direction*
  - *random mob objects falling*
  - *varying time due to health status...*
- underlying motor skills, for example, are the same for the player's character
  - *but the end result has now been challenged and thrown into doubt*

Games and dramatic elements

a story and premise

- a **premise** becomes a wrapper or container for our game
  - *we may use to create a sense of context for such challenges, skills, and fun*
  - *a sense of story...*
- each game we design and develop will include such a *premise*
  - *might be a single concept or a detailed dramatic backdrop*
- our games will often leverage a few well-known dramatic elements
  - *help create a player's connection and interest in a game's formal elements*
- use *premise* to help identify the game's formal elements within a setting or a metaphor
- without a sense of context and setting
  - *we may abstract mechanics, gameplay, and skills too far*
  - *reducing sense of fun for our player*
- consider difference between an outline of initial game logic and the wrapper a *premise* provides

quick exercise

Consider the following metaphors,

> *The skies of his future began to darken*

> *Her voice is music to his ears*

> *The ballerina was a swan, gliding across the stage*

> *A heart of stone*

Choose two of the above metaphors, and consider the following:

- how might your chosen metaphors shape the premise and story of a game?
- how might the premise of this game influence mechanics and skills for characters?
- how may you use such skills to create challenges in the game?
- how do your chosen metaphors, and the inferred premise, wrap this game's formal elements?

Python and Pygame - Game Example I

health and status - intro

- may add a status bar for a player's health, lives, &c.
- then dynamically update it relative to a defined health value
  - *e.g. a percentage value we decrement per collision*
- current game only gives a player one chance to shoot and destroy mob objects
  - *in effect, player currently has one life*
  - *one player life is not expected for most shooter style game...*
- may now consider monitoring and updating the status of a player's health
  - *e.g. as they are hit by advancing mob objects*
- protect our player, and their ship, using a Star Trek style shield
  - *may offer full protection initially*
  - *then incrementally weaken with each hit from a mob object*
  - *weakens until it eventually fails at value 0*
- set a default for this shield in the `Player` class,

```
# set default health for our player - start at max 100% and then decrease...
self.stShield = 100
```

Python and Pygame - Game Example I

health and status - collisions and shields

- need to modify our logic for a mob collision to ensure we handle such objects better
  - *may now reflect a decrease in the player's shield, health...*
- instead of allowing a mob object to continue after it has collided with the player
  - *now need to remove it from the game window*
- if we don't update this boolean to `True`
  - *each mob object will simply continue to hit the player*
  - *hit registered as it moves, pixel by pixel, through the player's ship*
  - *single hit would quickly become compounded in the gameplay*
- update for this check, e.g.

```
# add check for collision - enemy and player sprites (True = hit object is now deleted fro
collisions = pygame.sprite.spritecollide(player, mob_sprites, True, pygame.sprite.collide_
```

- as our player may be hit by multiple mob objects
  - *also need to update our check from a simple conditional to a loop*
  - *check possible collisions...*

```
# check collisions with player's ship - decrease shield for each hit
for collision in collisions:
    # decrease player's shield for each collision
    player.stShield -= 20
    # check overall shield value - quit game if no shield
    if player.stShield <= 0:
        running = False
```

Python and Pygame - Game Example I

health and status - replace mob objects

- we still have an issue with losing mob objects
  - *if they collide with the player's ship*
  - *follows same underlying pattern as player's laser beam firing on mob objects*
- need to create a new object if it is removed after a collision
- a familiar pattern we may now abstract
  - *creation of mob objects to avoid repetition of code, e.g.*

```
# create a mob object
def createMob():
    mob = Mob()
    # add to game_sprites group to get object updated
    game_sprites.add(mob)
    # add to mob_sprites group - use for collision detection &c.
    mob_sprites.add(mob)
```

- simple abstracted function allows us to easily recreate our mob objects
  - *by creating a mob object*
  - *adding it to the overall group of* `game_sprites`
  - *then the specific group for the game's* `mob_sprites`
- then call this function if a mob object collides with a projectile, player's ship...
- also call this function when we initially create our new mob objects

```
# create a new mob object
createMob()
```

Python and Pygame - Game Example I

health and status - health status bar

- already defined a default maximum for our player's shield
  - *now start to output its value to the game window*
- we could simply output a numerical value
  - *as we did for the player's score*
- more interesting to show a graphical update for the status of a player's health
- define a new draw function to render a visual health bar for player's shield, e.g.

```
# draw a status bar for the player's health - percentage of health
def drawStatusBar(surface, x, y, health_pct):
    # defaults for status bar dimension
    BAR_WIDTH = 100
    BAR_HEIGHT = 10
    # use health as percentage to calculate fill for status bar
    bar_fill = (health / 100) * BAR_WIDTH
    # rectangles - outline of status bar &
    bar_rect = pygame.Rect(x, y, BAR_WIDTH, BAR_HEIGHT)
    fill_rect = pygame.Rect(x, y, bar_fill, BAR_HEIGHT)
    # draw health status bar to the game window - 1 specifies pixels for border width
    pygame.draw.rect(surface, GREEN, fill_rect)
    pygame.draw.rect(surface, WHITE, bar_rect, 1)
```

- function accepts four parameters, which allow us to define
  - *a surface for rendering*
  - *its x and y location in the game window*
  - *then update the status of the player's health*
- set a default width and height for the status bar
  - *then specify how much of this bar needs to be filled with colour*
  - *colour fill relative to the player's current health status...*
- health status can be calculated as a percentage
  - *allows us to easily modify the relative sizes for the status bar*

resources

- notes = player-health-intro.pdf

- code = playerhealth1.py

Python and Pygame - Game Example 1

fun game extras - intro

- now start to add some fun extras to the general gameplay
  - *help improve the general player experience*
- a few examples
  - *modify health status bar to better reflect health percentages*
  - *auto fire for the laser beam to continuously shoot using space bar*
  - *fun explosions for collisions*
- many more...

Python and Pygame - Game Example 1

fun game extras - update health status colours

- modify health status bar to more accurately inform player of ship's health
- common option is to simply modify colour of status bar to reflect health status
- we may use a bright colour to indicate greater health status
  - *then change it to RED as a warning to the player, e.g.*

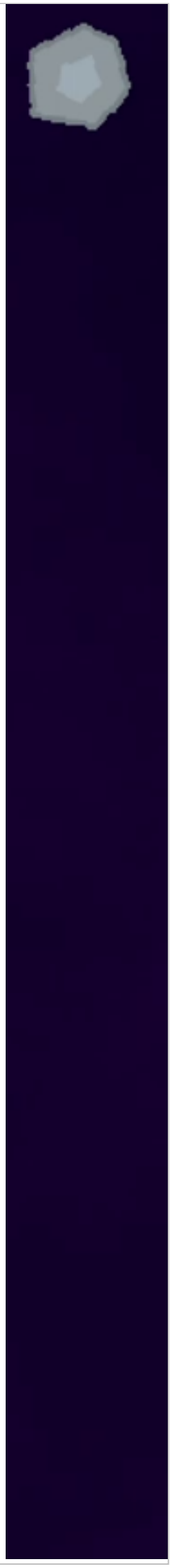
```
if bar_fill < 40:
    pygame.draw.rect(surface, RED, fill_rect)
else:
    pygame.draw.rect(surface, CYAN, fill_rect)
```

game example

- shooter1.0.py
- check player's health
  - *set default health to 100%*
  - *decrement health per collision*
    - quit game when health reaches 0
  - *draw status bar to game window*
    - green colour for good health
    - change to red colour below 40%

Video - Shooter 1.0

check player's health



Python and Pygame - Game Example I

fun game extras - repetitive firing sequence - intro

- add a repetitive firing sequence for the player's sprite object
- in our current game logic
  - *as a player presses down on the space bar a laser beam will be fired from the top of the player's ship*
  - *one press is equal to one firing sequence...*
- to add a repetitive firing sequence
  - *need to still check that the spacebar has been pressed down*
  - *but now continue to fire a laser beam until the key is released*
- in our `Player` class we can add some new variables, e.g.
  - *specify the delay in milliseconds between each firing of the laser beam*
  - *check the time, the number of ticks, since the last beam was fired*
  - *e.g. update `Player` class as follows,*

```
...
# firing delay between laser beams
self.firing_delay = 200
# time in ms since last fired
self.last_fired = pygame.time.get_ticks()
```

Python and Pygame - Game Example 1

fun game extras - repetitive firing sequence - fire - part 1

- add a listener for the space bar event to the `update()` method in the `Player` class

```
# check space bar for firing projectile
if key_state[pygame.K_SPACE]:
    # fire laser beam
    self.fire()
```

- update our `fire()` method to reflect this repetitive firring sequence, e.g.

```
...
# get current time
time_now = pygame.time.get_ticks()
if time_now - self.last_fired > self.firing_delay:
    self.last_fired = time_now
    ...
```

Python and Pygame - Game Example 1

fun game extras - repetitive firing sequence - fire - part 2

- our `fire()` method has now been updated as follows,

```
# fire projectile from top of player sprite object
def fire(self):
    # get current time
    time_now = pygame.time.get_ticks()
    if time_now - self.last_fired > self.firing_delay:
        self.last_fired = time_now
        # set position of projectile relative to player's object rect for centerx and top
        projectile = Projectile(self.rect.centerx, self.rect.top)
        # add projectile to game sprites group
        game_sprites.add(projectile)
        # add each projectile to sprite group for all projectiles
        projectiles.add(projectile)
        # play laser beam sound effect
        laser_effect.play()
```

- remove listener for a space bar event in the events section of the game loop

resources

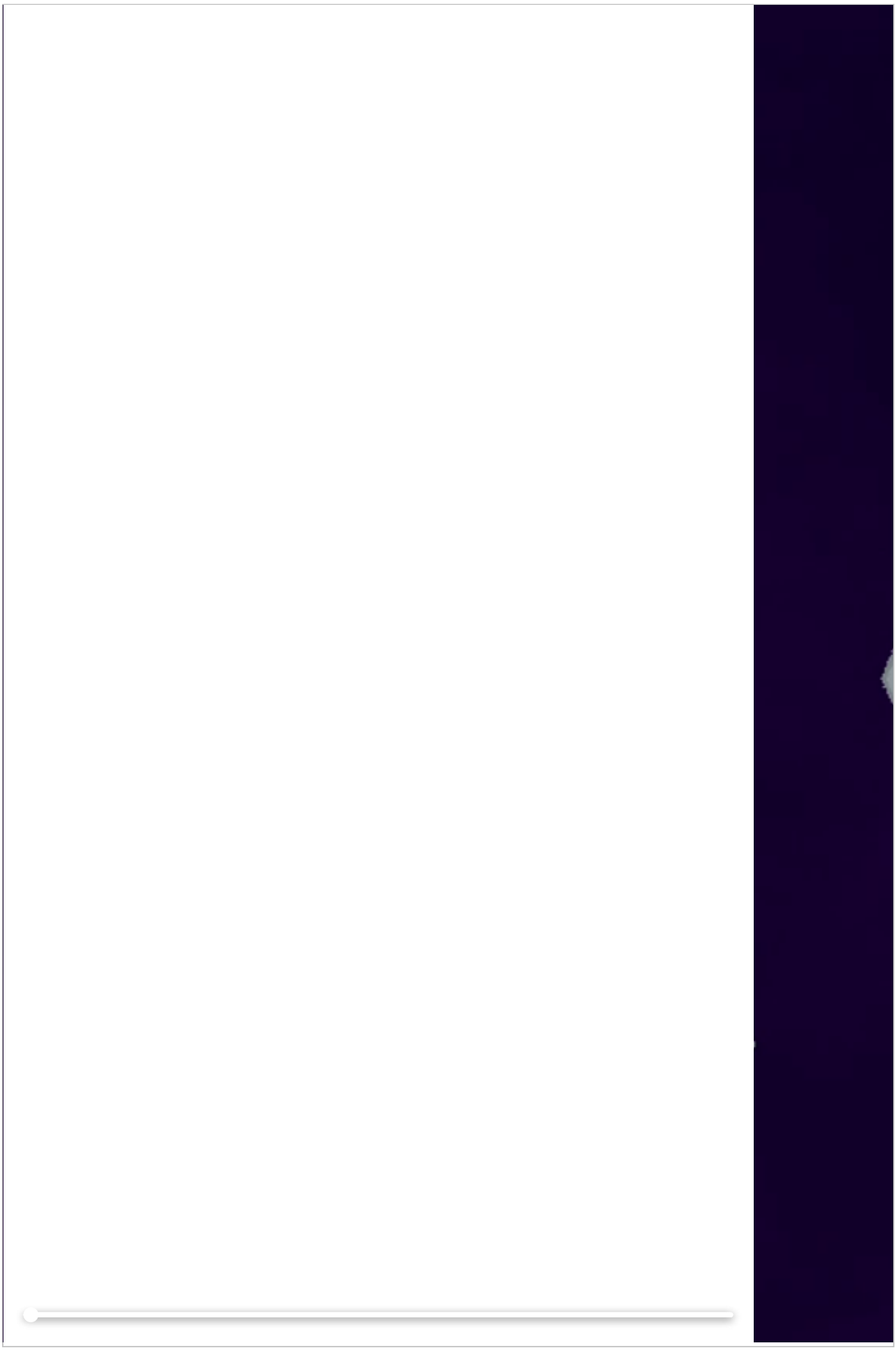
- notes = extras-part1-firing.pdf
- code = repetitivefiring.py

game example

- shooter1.1.py
- add repetitive firing sequence for player's laser beam
  - *move keypress check for space bar to player class*
  - *fire laser beam whilst space pressed down*
  - *set interval in ms for firing sequence*
  - *check time between now and last firing*

Video - Shooter 1.1

add repetitive firing sequence...



Demos

- pygame random sprites
  - *animatingsprites2.py*
- pygame drawing text
  - *drawingtext1.py (game with text)*
  - *drawingtext2.py (simple rendered text)*
- pygame music and sound effects
  - *basicmusic1.py*
  - *basicmusic2.py*
- pygame health and status
  - *playerhealth1.py*
- pygame extras
  - *repetitivefiring.py*
- pygame - Game 1 Example
  - *shooter0.6.py*
  - *shooter0.7.py*
  - *shooter0.8.py*
  - *shooter0.9.py*
  - *shooter1.0.py*
  - *shooter1.1.py*

Games

- Colin McRae Rally
- Diablo - Wikipedia
- Diablo III - console

Game notes

- Pygame
  - *sprites-intro.pdf*
  - *sprites-set-image.pdf*
  - *sprites-control.pdf*
  - *sprites-animating-images.pdf*
  - *sprites-animating-random-images.pdf*
  - *drawing-text.pdf*
  - *music-intro.pdf*
  - *player-health-intro.pdf*
  - *extras-part1-firing.pdf*

Resources

- Csikszentmihalyi, M. *Flow: The Psychology of Optimal Experience*. Harper & Row. New York. 1990.
- Various
  - *The Sims - Free Will*

Videos

- Colin McRae Rally - YouTube
- TED 2004 - Flow, the secret to happiness