

Comp 388/488 - Game Design and Development

Spring Semester 2019 - Week 7

Dr Nick Hayward

Python and Pygame - Game Example I

shooter style game - STG

- start creating our first full game example
 - *shooter example - **STG** in Japan*
- this game will help us design, develop, and test the following:
 - *user control*
 - *enemy objects*
 - *collision detection*
 - *firing projectiles at enemies*
 - *destroying enemy objects*
 - *add custom sprites and graphics*
 - *improve the collision detection*
 - *start animating sprite images*
 - *radomise enemy objects to create greater challenges*
 - *keep a running game score and render to game window*
 - *add game music and sound effects*
 - *check our player's health...*
 - *add some fun game extras*
 - *e.g. health status, explosions...*
 - *lots more...*

Python and Pygame - Game Example I

add more objects - mob

- now start to add extra sprite objects to our game window
 - *commonly given a collective, generic name of **mob***
- add the following class Mob to our game

```
# create a generic mob sprite for the game - standard name is *mob*
class Mob(pygame.sprite.Sprite):
    def __init__(self):
        pygame.sprite.Sprite.__init__(self)
        self.image = pygame.Surface((20, 20))
        self.image.fill(CYAN)
        # specify bounding rect for sprite
        self.rect = self.image.get_rect()
        # specify random start posn & speed of enemies
        self.rect.x = random.randrange(winWidth - self.rect.width)
        self.rect.y = random.randrange(-100, -50)
        self.speed_y = random.randrange(1, 10)

    def update(self):
        self.rect.y += self.speed_y
```

- with this class we can create extra sprite objects
 - *set their size, colour, &c.*
 - *then set random x and y coordinates for the starting position of the sprite object*
- use random values to ensure that the objects start and move from different positions
 - *from the top of the game window*
 - *then progress in staggered groups down the window...*

Python and Pygame - Game Example I

update extra objects

- as our enemy objects move down the game window
 - *need to check if and when they leave the bottom of the game window*
- we can add the following checks to the update function

```
def update(self):  
    self.rect.y += self.speed_y  
    # check if enemy sprite leaves the bottom of the game window - then randomise at t  
    if self.rect.top > windowHeight + 15:  
        # specify random start posn & speed of enemies  
        self.rect.x = random.randrange(winWidth - self.rect.width)  
        self.rect.y = random.randrange(-100, -50)  
        self.speed_y = random.randrange(1, 7)
```

- as each sprite object leaves the bottom of the game window
 - *we can check its position*
- then, we may reset the sprite object to the top of the game window
- need to ensure that the same sprite object does not simply loop around
 - *and then reappear at the same position at the top of the game window*
 - *becomes too easy and tedious for our player...*
- instead, we can reset our *mob* object to a random path down the window
 - *should make it slightly harder for our player*
- also ensure that each extra sprite object has a different speed
 - *by simply randomising the speed along the y-axis per sprite object*

Python and Pygame - Game Example I

show extra objects

- now create a *mob* group as a container for our extra sprite objects
- group will become particularly useful as we add collision detection later in the game
 - *update our code as follows, e.g.*

```
# sprite groups - game, mob...
mob_sprites = pygame.sprite.Group()
# create sprite objects, add to sprite groups...
for i in range(10):
    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)
```

- create our *mob* objects
 - *then add them to the required sprite groups*
- by adding them to the `game_sprites` group
 - *they will be updated as the game loop is executed*
- `mob_sprites` group will help us easily detect extra sprite objects
 - *e.g. when we need to add collision detection*
 - *or remove them from the game window...*

Python and Pygame - Game Example I

modify motion of extra objects - part I

- above updates work great for random motion along the y-axis
 - *add some variation to movement of extra sprite object by modifying the x-axis*
- we can modify the x-axis for each extra sprite object
 - *creates variant angular motion along both the x-axis and y-axis, e.g.*

```
# random speed along the x-axis
self.speed_x = random.randrange(-3, 3)
...

self.rect.x += self.speed_x
# check if sprite leaves the bottom of the game window - then randomise at the top...
if self.rect.top > winHeight + 15 or self.rect.left < -15 or self.rect.right > winWidth + 15:
    # specify random start posn & speed of extra sprite objects
    self.rect.x = random.randrange(winWidth - self.rect.width)
    self.speed_x = random.randrange(-3, 3)
...
```

Python and Pygame - Game Example I

modify motion of extra objects - part 2

- our mob class may now be updated as follows,

```
# create a generic extra sprite object for the game - standard name is *mob*
class Mob(pygame.sprite.Sprite):
    def __init__(self):
        pygame.sprite.Sprite.__init__(self)
        self.image = pygame.Surface((20, 20))
        self.image.fill(CYAN)
        # specify bounding rect for sprite
        self.rect = self.image.get_rect()
        # specify random start posn & speed
        self.rect.x = random.randrange(winWidth - self.rect.width)
        self.rect.y = random.randrange(-100, -50)
        # random speed along the x-axis
        self.speed_x = random.randrange(-3, 3)
        # random speed along the y-axis
        self.speed_y = random.randrange(1, 7)

    def update(self):
        self.rect.x += self.speed_x
        self.rect.y += self.speed_y
        # check if sprite leaves the bottom of the game window - then randomise at the top..
        if self.rect.top > windowHeight + 15 or self.rect.left < -15 or self.rect.right > winWi:
            # specify random start posn & speed of extra sprite objects
            self.rect.x = random.randrange(winWidth - self.rect.width)
            self.rect.y = random.randrange(-100, -50)
            self.speed_x = random.randrange(-3, 3)
            self.speed_y = random.randrange(1, 7)
```

- added a quick check for motion of our extra sprite object along the x-axis
 - as sprite exits on either side of the screen
 - create a new sprite on a random path down the screen

resources

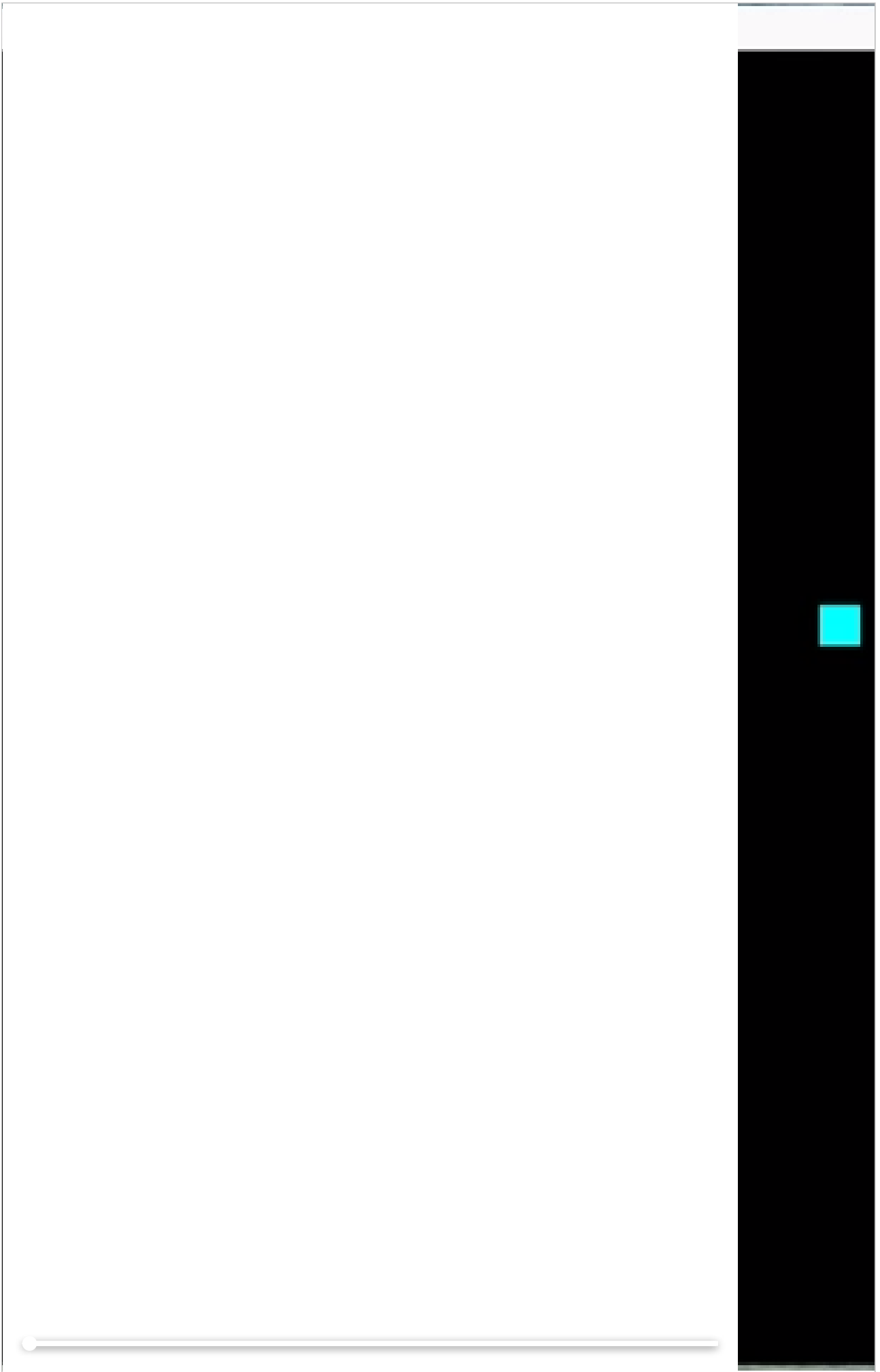
- notes = sprites-more-objects.pdf
- code = basicsprites4.py

game example

- shooter0.2.py
 - add enemy objects

Video - Shooter 0.2

move & control



Python and Pygame - Game Example I

add new sprites

- create a new class for this sprite object
 - e.g. *projectiles that a player may appear to fire from the top of player object*
 - *such as a ship &c*

```
# create a generic projectile sprite - for bullets, lasers &c.
class Projectile(pygame.sprite.Sprite):
    # x, y - add specific location for object relative to player sprite
    def __init__(self, x, y):
        pygame.sprite.Sprite.__init__(self)
        self.image = pygame.Surface((5, 10))
        self.image.fill(RED)
        self.rect = self.image.get_rect()
        # weapon fired from front (top) of player sprite...
        self.rect.bottom = y
        self.rect.centerx = x
        # speed of projectile up the screen
        self.speed_y = -10

    def update(self):
        # update y relative to speed of projectile on y-axis
        self.rect.y += self.speed_y
        # remove from game window - if it goes beyond bounding for y-axis at top...
        if self.rect.bottom < 0:
            # kill() removes specified sprite from group...
            self.kill()
```

- creating another sprite object for a projectile such as a bullet or a laser beam
- projectile will be shot from the top of another object
 - *set x and y coordinates relative to position of player's object*
 - *setting the speed along the y-axis so it travels up the screen*
- as we update each projectile object
 - *update its speed, and then check its position on the screen...*
- if it leaves the top of the game window
 - *we can call the generic `kill()` method on this sprite*
- method is available for any sprite object we create in the game window

Python and Pygame - Game Example I

listen for keypress

- need to add a new listener to the game loop to detect a keypress for the *spacebar*
- use this keypress to allow a player to shoot these projectiles, e.g. a laser beam

```
# 'processing' inputs (events)
for event in EVENTS.get():
    # check keyboard events - keydown
    if event.type == pygame.KEYDOWN:
        # check for ESCAPE key
        if event.key == pygame.K_ESCAPE:
            gameExit()
        elif event.key == pygame.K_SPACE:
            # fire laser beam...
            player.fire()
```

- updated our keypress listeners to check each time a player hits down on the *spacebar*
- use this keypress event to fire our projectile
 - e.g. a laser beam to hit our enemy mobs...

Python and Pygame - Game Example I

release new sprites

- as player hits the *spacebar*, we need to create new sprites
- new sprite objects will then be released from the top of the player's object
- relative position of one sprite object is determining start position of another sprite object
- need to update the class for our primary sprite object, e.g. a player
 - *include a method for firing the projectiles from the top of this sprite object, e.g.*

```
# fire projectile from top of player sprite object
def fire(self):
    # 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)
```

- sets start position for x and y coordinates of each projectile sprite
 - *sets to the current position of the player's sprite object*
- then, add each projectile sprite object to the main game sprite group
 - *and add a new sprite group for all of the projectiles*
 - *add this new sprite group as follows,*

```
projectiles = pygame.sprite.Group()
```

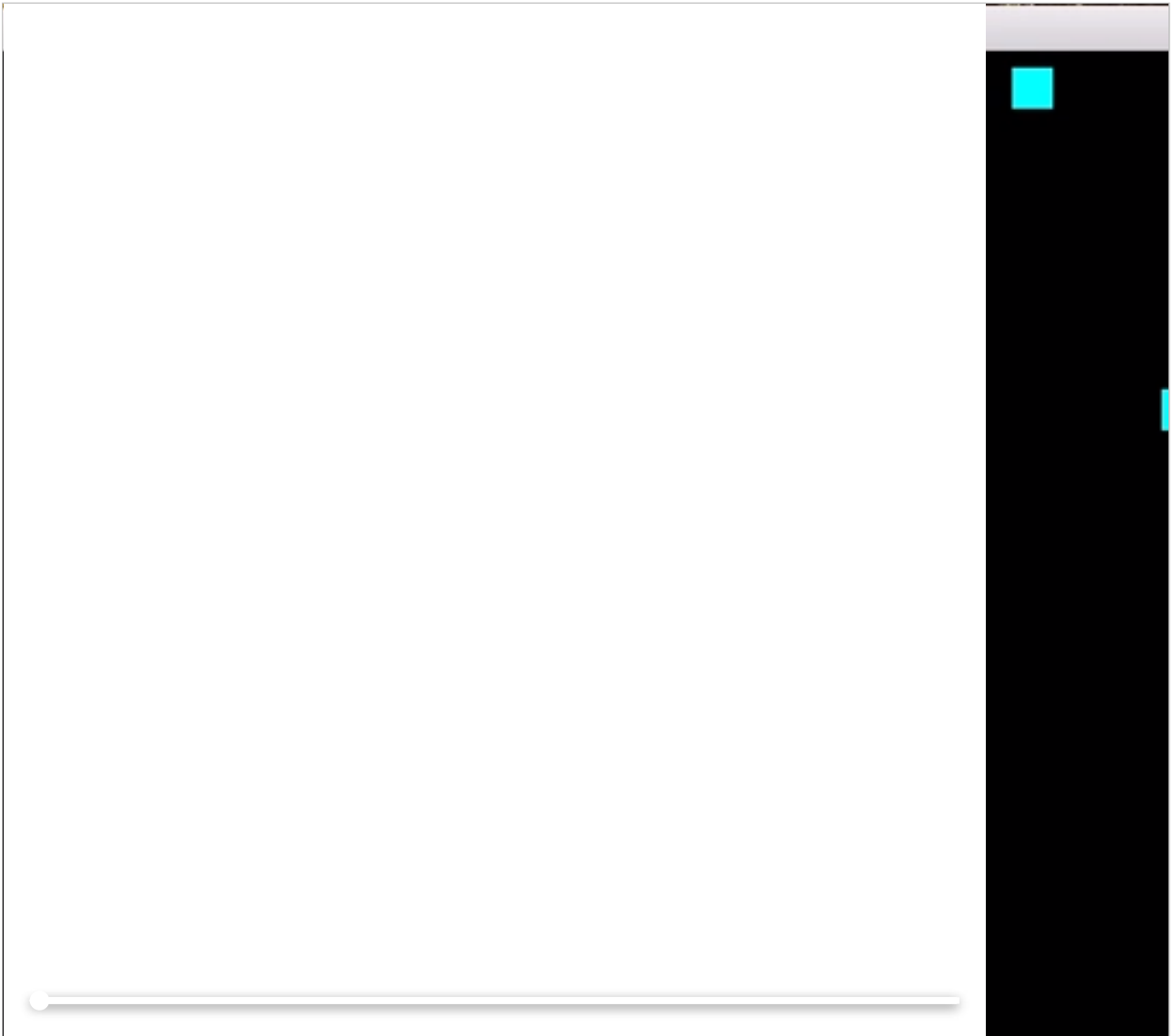
- when a player presses down on the *spacebar* a projectile will be fired
 - *a red laser beam from the top of the player's sprite object*

resources

- notes = sprites-relative-objects.pdf
- code = basicsprites5.py

Video - Basic Sprites

relative objects



Python and Pygame - Game Example I

basic collision detection

- Pygame includes support for adding explicit collision detection
 - *between two or more sprites in a game window*
 - *use built-in functions to help us work with these collisions*
- add basic collision detection
 - *each time an object hits the player's object at the foot of the game window*
 - *Pygame includes the following function, e.g.*

```
# add check for collision - extra objects and player sprites (False = hit object is not deleted)
pygame.sprite.spritecollide(player, mob_sprites, False)
```

- `sprite` object's function allows us to check if one `sprite` object has been hit by another
- e.g. checking if `player` `sprite` object hit by another `sprite` object
 - *in this example, from the `mob_sprites` group*
- `False` parameter is a boolean value for the state of the object that has hit
 - *i.e. determines whether a `mob` `sprite` object should be deleted from game window or not*
- particularly useful as it returns a `list` data structure
 - *contains any `mob` `sprite` objects that hit the `player` `sprite` object*
 - *update this code as follows, and store this list in a variable, e.g.*

```
collisions = pygame.sprite.spritecollide(player, mob_sprites, False)
```

- then use this `list` to check if any collisions have occurred in our game window, e.g.

```
...
if collisions:
    # update game objects &c.
    ...
...
```

- use boolean value to check if the `list` `collisions` is empty or not

Python and Pygame - Game Example I

Sprite group collision detection

- now add collision detection for various groups of sprites
 - e.g. one group of sprites may be colliding with another, defined sprite group...
- use Pygame's collide method for sprite groups, e.g.

```
# add check for sprite group collide with another sprite group - projectiles hitting enemy
collisions = pygame.sprite.groupcollide(mob_sprites, projectiles, True, True)
```

- boolean parameter values of True and True
 - allow us to delete both the hit enemy objects
 - and the projectile objects that hit them
- as list of collisions is populated
- create new sprite objects for those that have been hit and deleted
- e.g. extra objects that move down the game window

```
# add more mobs for those hit and deleted by projectiles
for collision in collisions:
    mob = Mob()
    game_sprites.add(mob)
    mob_sprites.add(mob)
```

- if we don't create new extra objects
 - game window will quickly run out of sprite objects

resources

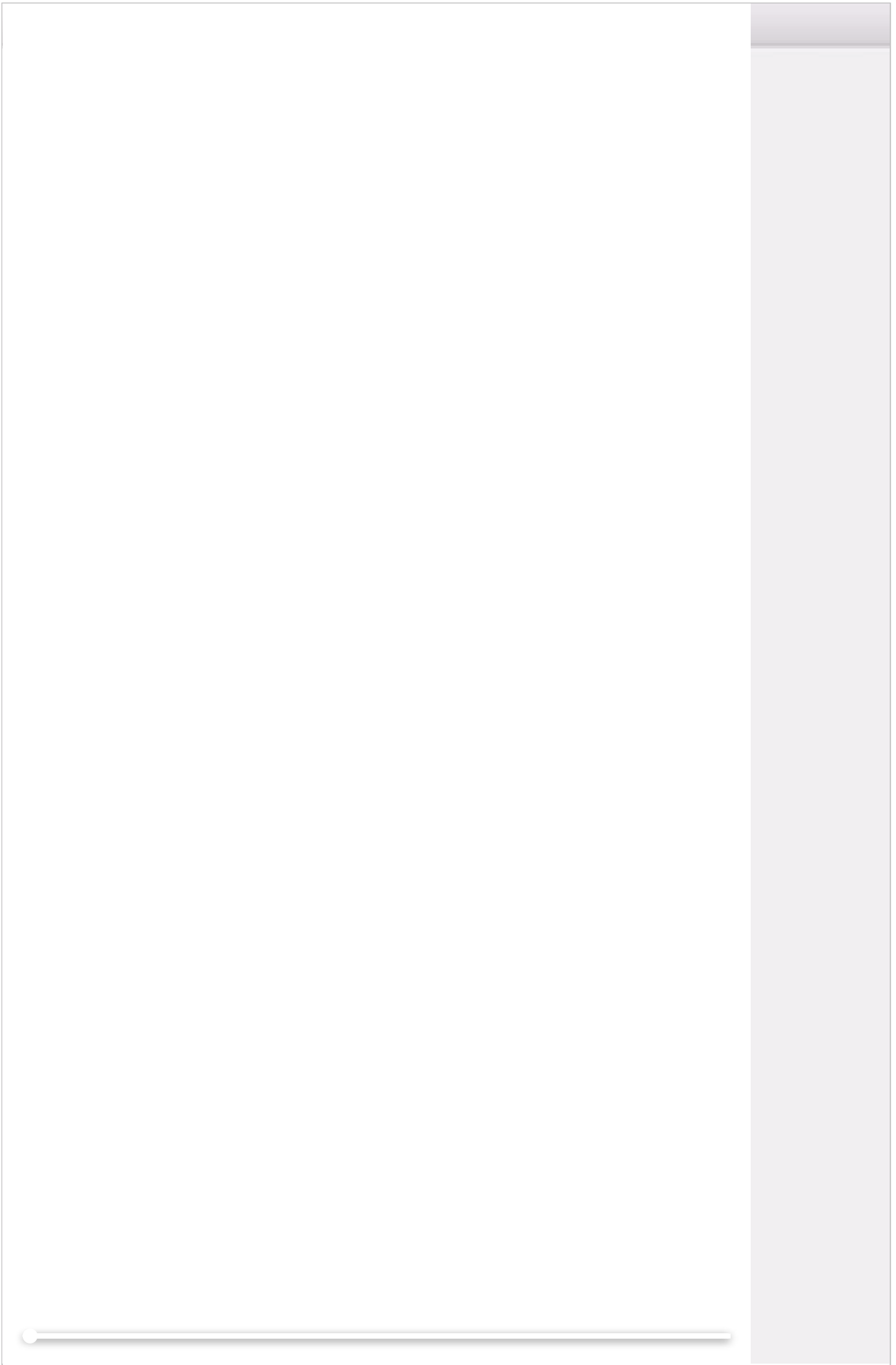
- notes = sprites-collision-detection.pdf
- code = basicsprites6.py

game example

- shooter0.3.py
 - collision detection of single sprite
 - detect group collisions

Video - Shooter 0.3

basic collisions and firing



Python and Pygame - Game Example I

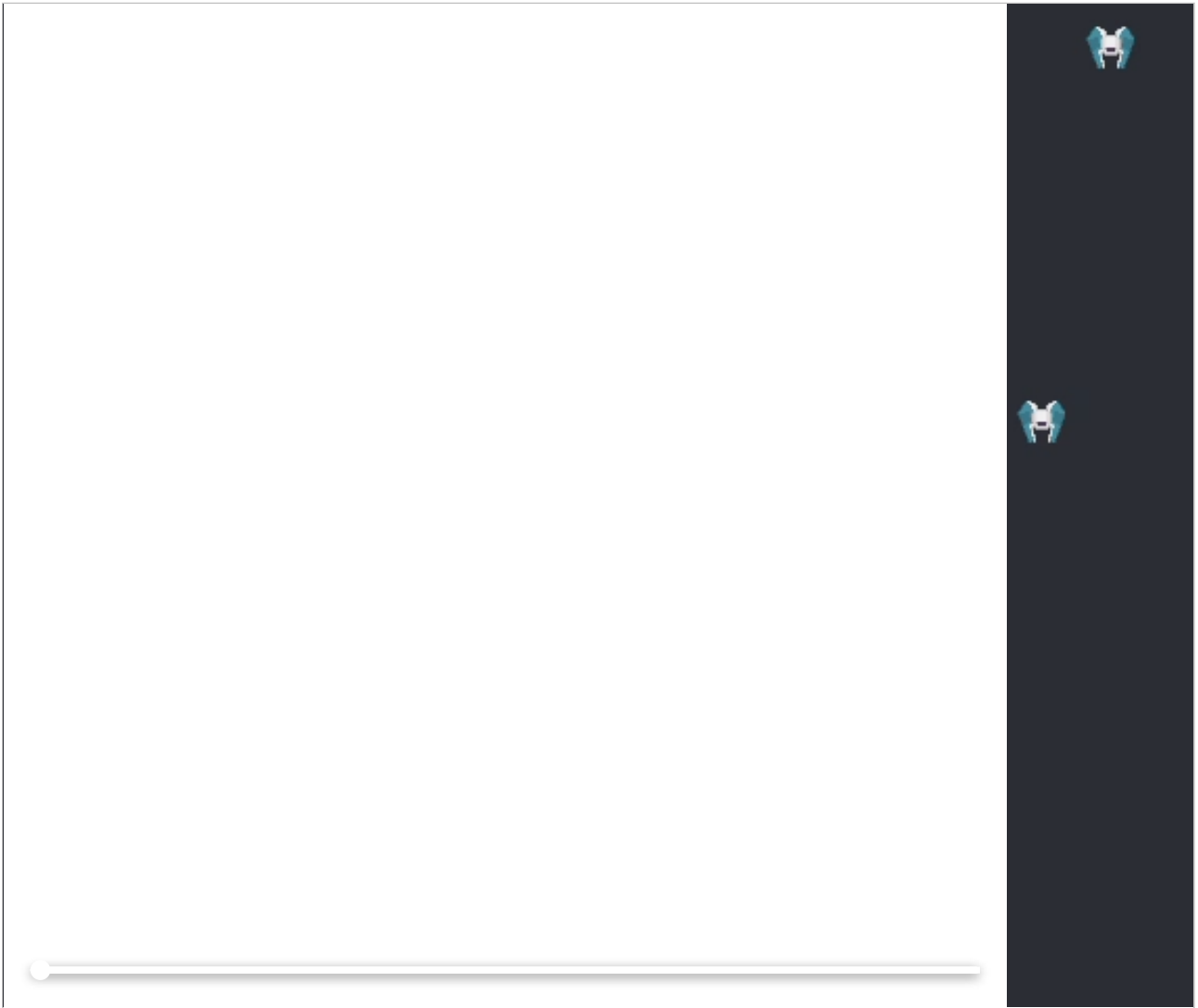
add graphics to the sprites

- now start to add some custom images for our sprite objects
 - *player object, mobs, projectiles, and a game background...*
- add images and backgrounds to our shooter game to help represent objects
 - *player's ship, laser beams firing, asteroids to hit, and star-filled background*
- before we can add our images for the sprites and backgrounds
 - *need to add some images files to our game's directory structure*
 - *normally create an `assets` folder*
 - *add any required images, audio, video &c. for our game...*
- may now update our directory structure to include the required assets,

```
|-- shootemup
    |-- assets
        |-- images
            |__ ship.png
```

Video - Add Graphics

add images to the game



Python and Pygame - Game Example I

import game assets

- need to import the Python module for `os`
- allows us to query a local OS's directory structure.

```
# import os
import os
```

- specify the directory location of the main game file
 - so Python can keep track of the relative location of this file, e.g.

```
game_dir = os.path.dirname(__file__)
```

- `__file__` is used by Python to abstract the root application file
 - then portable from system to system
 - allows us to set relative paths for game directories, e.g.

```
# game assets
game_dir = os.path.dirname(__file__)
# relative path to assets dir
assets_dir = os.path.join(game_dir, "assets")
# relative path to image dir
img_dir = os.path.join(assets_dir, "images")
```

- may then import an image for use as a sprite as follows,

```
# assets - images
ship = pygame.image.load(os.path.join(img_dir, "ship.png"))
```

Python and Pygame - Game Example I

convert and colour key

- as we import an image for use as a sprite within our game
 - *need to use a `convert()` method*
 - *ensures image file is of a type Pygame can use natively*
- if not, there is a potential for the game to perform more slowly
- convert example,

```
ship = pygame.image.load(os.path.join(img_dir, "ship.png")).convert()
```

- for each image that Pygame adds as a sprite
 - *a bounding rectangle will be set with a given colour*
- in most examples, we want to set the background of our sprite to transparent
- rectangle for the image will now blend with the background colour of our game window, e.g.

```
ball.set_colorkey(WHITE)
```

- now check for white coloured pixels in the image background
 - *then set them to transparent*

Python and Pygame - Game Example I

add game background

- now add a background image for our game
 - *we might recreate stars and space for our game window, e.g.*

```
# load graphics
bg_img = pygame.image.load(os.path.join(img_dir, "bg-purple.png")).convert
```

- also add a rectangle to contain our background image

```
# add rect for bg - helps locate background
bg_rect = bg_img.get_rect()
```

- basically helps us know where to add our background image
 - *then subsequently find it as needed with the logic of our game*
 - *then draw our background image as part of the game loop, e.g.*

```
# draw background image - specify image file and rect to load image
window.blit(bg_img, bg_rect)
```

Python and Pygame - Game Example I

add game images

- need to add an image for our player's ship, laser beams, and asteroids to shoot, e.g.

```
# add ship image
ship_img = pygame.image.load(os.path.join(img_dir, "ship-blue.png")).convert()
# ship's laser
laser_img = pygame.image.load(os.path.join(img_dir, "laser-blue.png")).convert()
# asteroid
asteroid_img = pygame.image.load(os.path.join(img_dir, "asteroid-med-grey.png")).convert()
```

- to use these new images in our game
 - need to modify the code for each object, e.g. *Player* object
 - update our class to include a reference to the *ship_img*

```
self.image = ship_img
```

- also customise this image by scaling it to better fit our game window, e.g.

```
# load ship image & scale to fit game window...
self.image = pygame.transform.scale(ship_img, (49, 37))
# set colorkey to remove black background for ship's rect
self.image.set_colorkey(BLACK)
```

- also update our ship's rect using a colorkey
 - ensures black rect is not visible in the game window

resources

- notes = graphics-and-sprites.pdf
- code = graphicssprites1.py

game example

- shooter0.4.py
 - add graphics for sprites
 - images for player's ship, ship's laser, and asteroids &c.
 - set colorkey for rect of sprite's
 - set background image for game window...

Video - Shooter 0.4

add graphics for sprites



Game Dev resources

music, sound effects...

- for a game's sound effects
 - *many different options and sources for these sounds*
- try open source examples, e.g.
 - *Open Game Art*
- perhaps create our own custom sounds using a utility such as **SFXR**, e.g.
 - *SFXR*
- or its derivative website option, e.g.
 - *BFXR*

Games and formal elements

intro

- as with each formal structure
 - *players*
 - *objectives*
 - *procedures & rules*
 - including implied **boundaries**
 - *conflict, challenge, battle...*
 - *outcome, end result...*
- these constituent elements come together
 - *to form what we largely understand to be a game*
- such formal elements constitute how we
 - *design*
 - *structure*
 - *develop our video games*
- overlap and interplay of these formal elements
 - *has now become the foundation for game design*
- a sound understanding and knowledge of these formal elements
 - *their usage and application*
 - *helps us start creating innovative, playful game experiences*

Games and formal elements

players and games - part I

- identified the need for rules, procedures...
- within the confines of such rules
 - *players are suspending normal societal restrictions*
 - *players enact shooting, fighting, role-play, and magical roles...*
 - *roles, actions &c. normally confined to a passive medium, e.g. books, film...*
- such actions can often form stark contrasts in a game environment
- rules become enacted in a **magic circle**
 - *originally described by Huizinga in his 1938 title, **Homo Ludens***
 - *later adapted, and refashioned for digital games*

In a very basic sense, the magic circle of a game is where the game takes place. To play a game means entering into a magic circle, or perhaps creating one as a game begins.




Salen, K. & Zimmerman, E. *Rules of Play: Game Design Fundamentals*. MIT Press. 2003.

Games and formal elements

players and games - part 2

- such rules naturally create the opportunity for play
 - *within the defined confines of a game*
- our use of rules, characters, story, and even mechanics and control
 - *invites a player to become involved with, and invested in, our game*
- motion controllers became an invitation for players to intuitively enter a game
 - *e.g. Nintendo's Wii, Microsoft's Xbox Kinect, and Sony's Playstation Move...*
- the premise of many games now became an extension of the controller
- it's not only a matter of engaging and inviting players into your game
- need to consider the nature and structure of player participation, e.g.
 - *how many players does the game support?*
 - *will each player adopt the same role?*
 - *perhaps play in a team or in direct competition*
- how we answer such questions will have a direct influence
 - *on the nature of the game we're designing*
 - *its gameplay*
 - *and a player's engagement with the story and characters...*

Image - Motion Controllers

Nintendo Wii	Xbox Kinect	Playstation Move
		

Video - Xbox Kinect & Algorithms



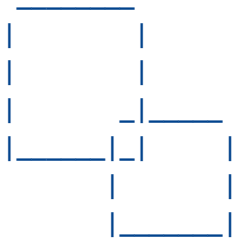
Kinect algorithm starts at 47:36 minutes into the video.

Source - Algorithms, YouTube

Python and Pygame - Game Example I

better collision detection

- collision detection is currently using rectangles to detect one sprite colliding with another
 - *technically referred to as **Axis-aligned Bounding Box** (AABB)*
- for some sprite images this will often cause an unrealistic effect as the two images collide
 - *image does not appear to collide with the other image*
 - *due to space caused by each respective rectangle*
- as one corner of a rectangle hits another corner a collision will be detected, e.g.



- unless each sprites image fits exactly inside the respective bounding box
 - *there will be space left over...*
- a few options for rectifying this issue
 - *might choose to simply calculate and set a slightly smaller rectangle*
 - *or, use a circular bounding box for our sprite image*
- benefit of an *axis-aligned bounding box*
 - *game is able to detect and calculate collisions much faster for a rectangle bounding box*
- a *circular bounding box* may be slower
 - *simply due to the number of calculations the game may need to perform*
 - *checks radius of one bounding box against another bounding box radius*
- rarely becomes a practical issue
 - *unless you're trying to work with thousands of potential sprite images*
- another option, the most precise as well
 - *use pixel perfect collision detection (PPCD)*

- PPCD - game engine will check each pixel of each possible sprite image
 - *determines if and when they collided*
 - *particularly resource intensive unless you require such precision*

Python and Pygame - Game Example I

add circle bounding box - part I

- add some *circle bounding boxes* to our sprite images
 - for *player* and *mob* objects
- start by adding explicit circles with a fill colour
 - helps us check the relative position of the circle's bounding box, e.g.

```
self.radius = 20
pygame.draw.circle(self.image, RED, self.rect.center, self.radius)
```

- we know sprite image for player's object will have a fixed, known size
 - we may set the radius to 20
- we may add some *circle bounding boxes* to the mob objects as well, e.g.

```
self.radius = int(self.rect.width * 0.9 / 2)
pygame.draw.circle(self.image, RED, self.rect.center, self.radius)
```

Python and Pygame - Game Example I

add circle bounding box - part 2

- used same basic pattern to add circles
 - *for mob objects we may set each circle's radius relative to the sprite image*
 - *setting radius as 90% of width of sprite image*
 - *then returning half of that value...*
- to use each *circle bounding box*, we need to update the collision checks as well
 - *update this check for each mob object in the update section of the game loop, e.g.*

```
# add check for collision - enemy and player sprites (False = hit object is not deleted fr  
collisions = pygame.sprite.spritecollide(player, mob_sprites, False, pygame.sprite.collide
```

- updated the collision check to explicitly look for circle collisions
 - *now remove explicit drawn circle for each circle bounding box*
 - *e.g. for the player and mob object sprites*
 - *we may simply comment out the drawn circle*

```
self.radius = int(self.rect.width * 0.9 / 2)  
#pygame.draw.circle(self.image, RED, self.rect.center, self.radius)
```

resources

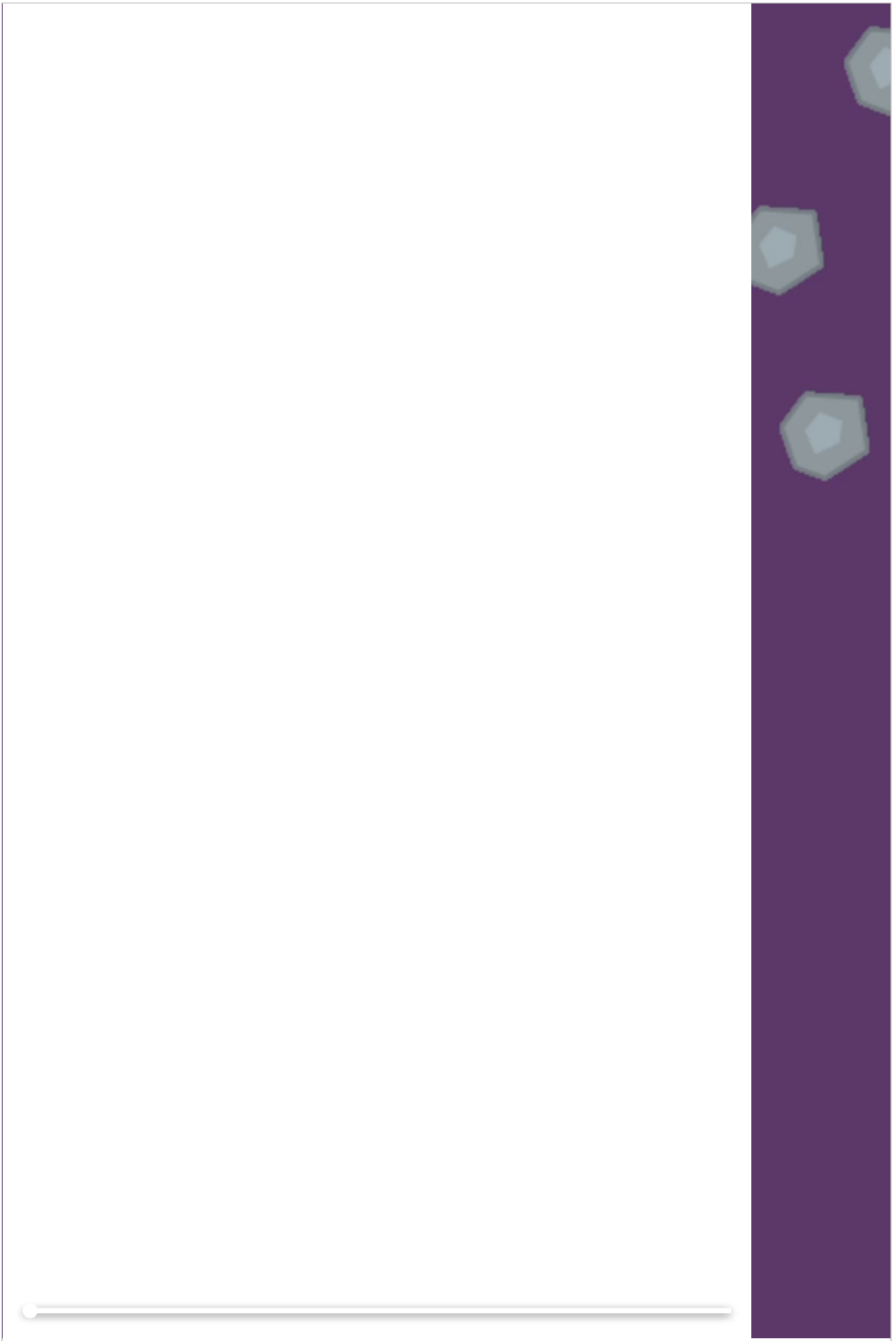
- notes = sprites-collision-detection-better.pdf
- code = collisionsprites3.py

game example

- shooter0.5.py
 - *better collisions and detection*
 - change bounding box for player and mob sprite objects
 - change bounding box to circle, and modify radius to fit sprite objects

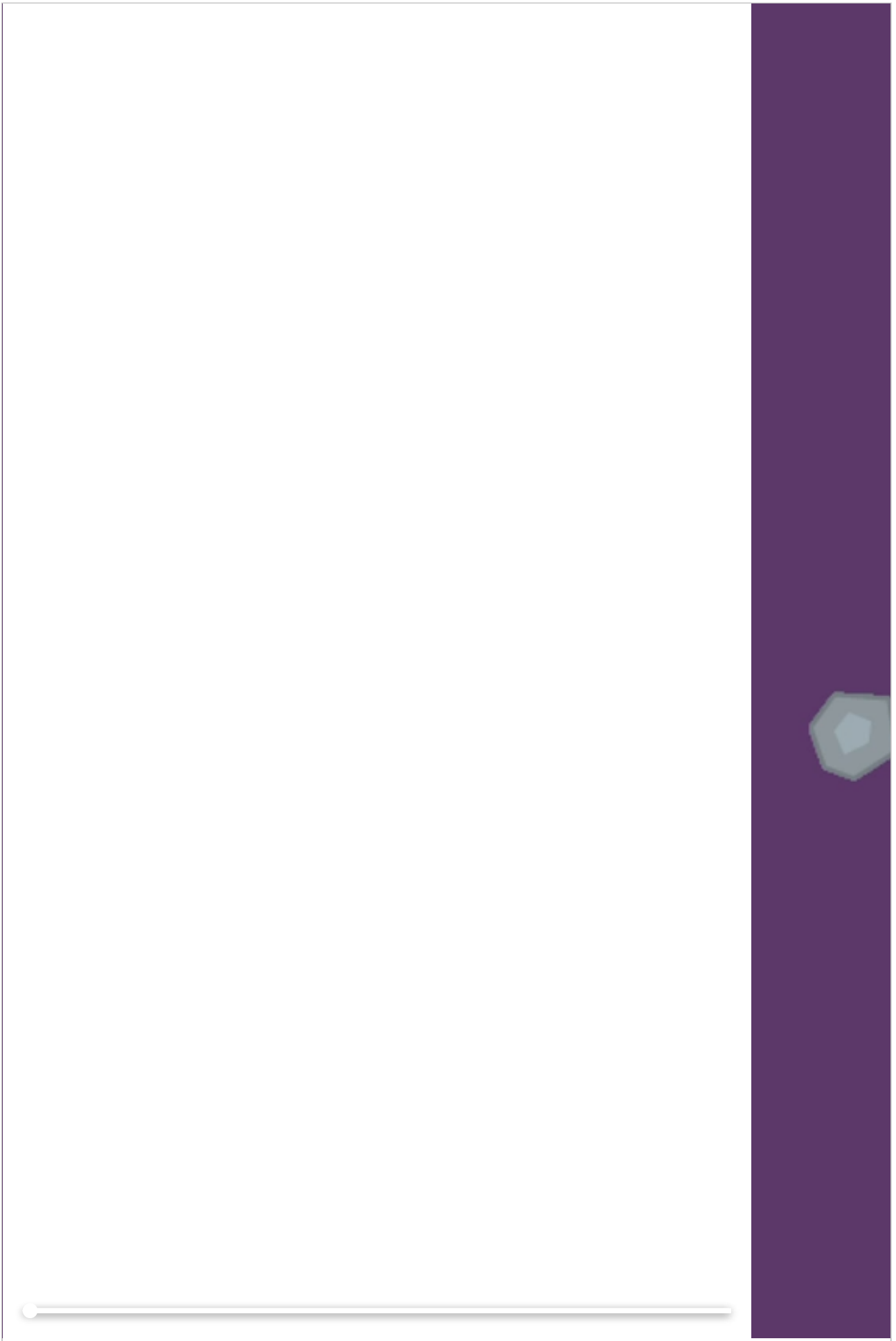
Video - Shooter 0.5

better collision detection - example I



Video - Shooter 0.5

better collision detection - example 2



Games and formal elements

players, patterns, and numbers

- games may be designed and developed for a variety of player numbers
 - *from strict single player options to varied multiplayer environments*
 - *MMOG push player numbers to the upper bounds*
- player numbers will often determine patterns of interaction for your game
- patterns may include
 - *single player versus the game*
 - e.g. Space Invaders, Mario, Sonic...
 - *multiple individual players versus the game*
 - e.g. sports, racing, card games...
 - *single player versus another player*
 - e.g. games such as Street Fighter and Mortal Kombat...
 - *multiple players versus a single player*
 - many detective and role playing board games include such features
 - some *god* games may also be structured using this pattern
 - *collaborative play*
 - players work together against the game
 - sports games, *Journey* by designer Jenova Chen...
 - *multiple players competing*
 - e.g. Halo, Call of Duty...
 - standard pattern referenced for *multiplayer* games
 - *team competitions...*
 - e.g. eSports such as League of Legends...

Games and Dynamics

systems and evolution - intro

- a game's players, their type, number, reactions, behaviours &c.
 - *may also be a reflection of the game system itself*
- systems may often display complex and unpredictable results
 - *e.g. when set in motion as part of the broader, general gameplay*
- such systems are not inherently predicated on complexity and scope
- good examples of simple rule sets and patterns
 - *produce unpredictable results as they are set in motion*
- we may see such patterns on a regular basis, e.g. in the natural world
 - *complex systems emerging due to collaborative structures, e.g.*
 - *ant colony*
 - *bee hive and pollen collection*
 - *swarm intelligence of a confusion of wildebeest*
- human consciousness may be the product of such systems
 - *commonly referred to as emergent systems*
- well-known experiment in emergence was conducted by John Conway in the 1960s
 - *particularly useful and interesting for us as game designers and developers*
- Conway was particularly curious about the working systems of rudimentary elements
 - *how did such elements work together based upon a set of defined, simple rules...*
- he wanted to clearly demonstrate this phenomenon at its simplest level
 - *e.g. in a defined 2D space, such as a known chess board*
- he tested various ideas and concepts
 - *considered ideas such as on and off logic for squares/cells on a board*
 - *logic based on rules for adjacent squares/cells*
- he continued his experiments and tests
 - *in a similar manner to a game designer*
 - *toyed with various sets of rules for several years*

Games and Dynamics

systems and evolution - simple rules

rules

- Birth
 - *if a cell is unpopulated, and surrounded by exactly 3 populated cells, this cell will be populated in the next generation*
- Death by loneliness
 - *if a cell is populated, and surrounded by fewer than 2 populated cells, this cell will be unpopulated in the next generation*
- Death by overpopulation
 - *if a cell is populated, and surrounded by at least 4 populated cells, this cell will be unpopulated in the next generation*
- emergent system finally converged on the above set of rules
- Conway, and some of his colleagues at Cambridge, began populating their chess board with pieces
 - *then tested their rules by hand*
- started to learn about this system
 - *and the very nature of emergent, almost evolving systems*
- quickly realised that different starting conditions had a noticeable impact on a system's evolution
- realised that the complexity of such start conditions might have a side-effect on the patterns created
 - *many simply failing to survive and evolve*
- a particularly interesting discovery became known as the **R Pentomino** configuration
 - *Richard Guy, an associate of Conway, became fascinated with this particular configuration*
- Guy tested their defined rules for more than a hundred generations
 - *started to observe various patterns emerging*
- a regular mix of shapes and patterns emerging
 - *Guy noticed that his shapes appeared to moving, effectively walking across the board*

- *he exclaimed at this discovery,*

Look, my bit's walking

- Poundstone, W. *Prisoner's Dilemma*. Touchstone. New York. 2002.
- Guy continued to test and work on this configuration
 - *until he was able to get this pattern to actually walk across the room*
 - *and then out the door..*
- Guy's discovery became known simply as a **glider**

Image - Systems and Evolution

R-Pentomino

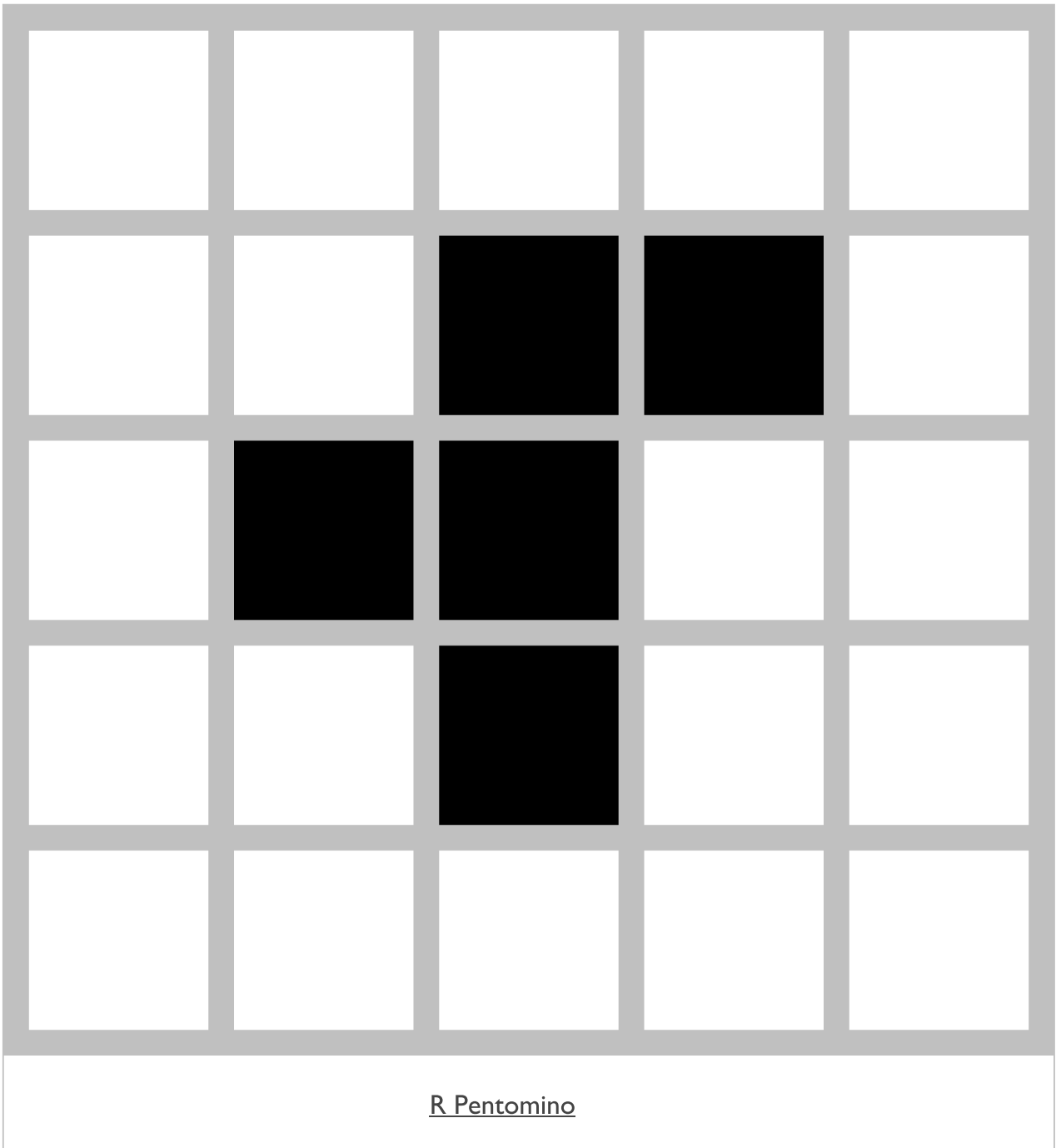
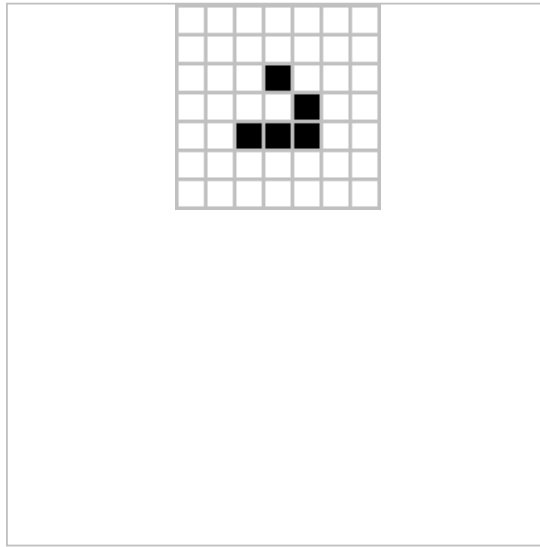


Image - Systems and Evolution

glider evolution



- [interactive demo - glider](#)

Games and Dynamics

systems and evolution - examples

- such simple emergent systems demonstrated
 - *benefits and application of simplicity in rules and patterns*
 - *their ability to evolve into life-form style patterns*
- such systems had the potential to evolve and develop with each generation
 - *particularly interesting and useful to us as game designers*
- may start to add such techniques to help make our games
 - *more realistic, evolving, and unpredictable for our players...*
- example games using these techniques include:
 - *Black and White*
 - *Grand Theft Auto (v3 onwards)*
 - *Halo*
 - *Oddworld: Munch's Oddysee*
 - *The Sims*
 - ...

Games and development

quick exercise

A quick exercise to consider evolution in systems,

- *Traveling Salesman Problem*
- evolution of simple systems
- swarm/hive intelligence
- interaction in systems

Then,

- consider the above, and how they might interact in a system to evolve an optimal solution to a problem
- consider application of such simple systems and evolution in a game environment

Video - Algorithms and Evolution



Algorithms and evolution starts at 31:20 minutes into the video.

Source - Algorithms, YouTube

Demos

- pygame collision detection - basic
- collisionsprites1.py
- collisionsprites2.py
- pygame collision detection - better
- collisionsprites3.py
- pygame sprites
- basicsprites4.py
- basicsprites5.py
- basicsprites6.py
- graphicssprites1.py
- pygame - Game 1 Example
- shooter0.1.py
- shooter0.2.py
- shooter0.3.py
- shooter0.4.py
- shooter0.5.py

Game notes

- Pygame
- [sprites-more-objects.pdf](#)
- [sprites-relative-objects.pdf](#)
- [sprites-collision-detection.pdf](#)
- [graphics-and-sprites.pdf](#)
- [sprites-collision-detection-better.pdf](#)

Resources

- Huizinga, J. *Homo Ludens: A Study of the Play-Element in Culture*. Angelico Press. 2016.
- Poundstone, W. *Prisoner's Dilemma*. Touchstone. New York. 2002.
- Salen, K. & Zimmerman, E. *Rules of Play: Game Design Fundamentals*. MIT Press. 2003.
- Conway and Life Patterns
- LifeWiki
- Richard Guy
- Glider
- Various
- BFXR
- Homo Ludens
- Open Game Art
- SFXR

Videos

- Algorithms - YouTube