Game Studies: studying the game w.r.t. its players, design … and their role in the society.

Game Theory: A mathematical method of decision making of a competitive situation (game) that is analyzed to determine the optimal course of an action for an interested party (agent).

Game studies is a huge field:

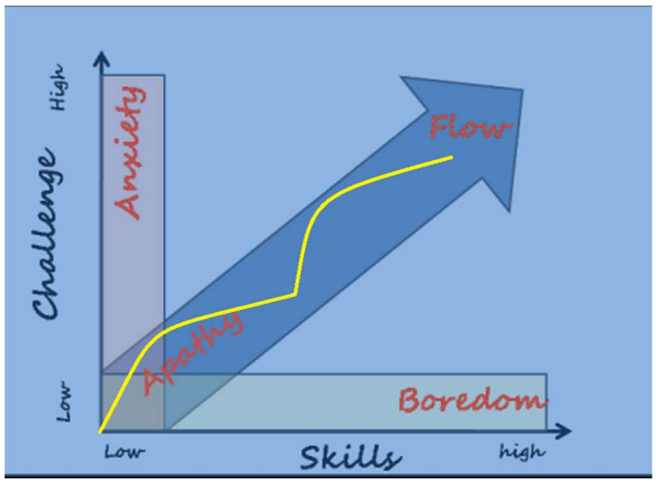
1. **Sociologist View (Psychology):** effect on people
2. **Dynamics View (Structure):** game's rules
3. **Engineering View (Industry):** how to make better games/design/development.

Game Funny? (1st View)

1. **Spatial Reasoning (Physical):** searching for 3D objects that interact with body, eye, hands..
2. **Pattern Recognition (Mental):** candy crush game, cube …
3. **Social:** communication skills, competition, coordination with others … (social situation).

** We give "real" emotions/psychological responses although we know the game is not "real"; (anger, anxiety, fear, pride..) => **Immersive** and **Flowing** gameplay.

Flow: deep sense of enjoyment so we keep the player **immersed** in the gameplay, it starts with **low level of challenge** to gain the **starting skills,** and **then it increases gradually** to avoid boredom.



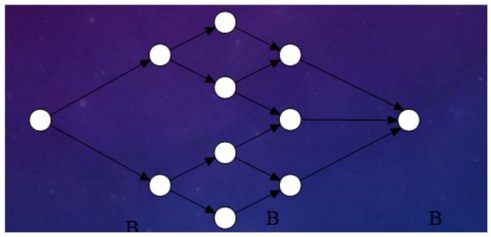
Structure View (2nd View): objects, rules, stories, objectives …

1. **Ludology/Ludologists:** focus on the games' rules.
2. **Narratology:**
   i) game should be understood as a novel of story-telling.
   ii) Adding narrative to games:
      a. Make it more compelling game (player is more convinced)
      b. May add "social component."
   iii) Narrative in games is like that in films (that what makes the player **immersed** and has **real emotion responses**):
      a. Don't cut the narrative plane **or** chain.
      b. Use the camera to immerse the viewer **or** to frame action.

iv) Narrative could be small cut scenes (all used for interactivity)

Concrete Things:

1. **Goals:** should be multiple simultaneous goals, achievable, and clear (always have at least one clear goal).
2. **Illusion of choice:** the player should think that his/her choice matter.
   --Convexity of choice: 1 choice -> widen to many choices -> return to single choice (result).



3. **Rewards and Punishments:** Value of an item depends on how much it costs the individual to acquire (money, effort, time …) -> spend more time and effort => **reward,** not achieving a goal/objective/task => **punishment**
   --Nowadays, players are **addicted to games** by **rewards and punishment (**punish if not playing, random rewards, loop reward/15 min …**)**

Engineering View (3rd View): keep interfaces simple (shortcuts for advanced players), provide feedback for EVERY action, design for error and special needs.

MDA: is a game development paradigm designed to help developers:

1. To make the most out of a game idea
2. To proceed efficiently through the complex process of bringing a game to market.

A. **Mechanics:** one should think before coding about: programming languages/ engines / tools/ libraries; the hardware required/ available; the software and hardware interfaces …
B. **Dynamics:** this is the **"ludological" part** and includes: the domain, rules, objects, players … of the game.
C. **Aesthetics (Decoration):** this is the **"narratological" part** and it is about the **"look of the game"** and includes: Colors, lighting, physical look of all players …

```
//Rotation, Scale
transform.Rotate(0, 0, Time.deltaTime);
transform.localScale += new Vector3(Time.deltaTime, Time.deltaTime, 0);

//Input Axis:
 transform.localScale += new Vector3(Time.deltaTime * Input.GetAxis("Fire1"), Time.deltaTime *
Input.GetAxis("Fire1"),0);

 transform.Translate (Time.deltaTime * Input.GetAxis("Horizontal") * 2.0f, Time.deltaTime *
Input.GetAxis("Vertical") * 2.0f,  0.0f);

 private void OnCollisionEnter2D(Collision2D other)

 private void OnTriggerEnter2D(Collider2D other)

 Input.GetKeyDown("space")

//Instantiation
 public GameObject pinPrefab;

Instantiate(pinPrefab, new Vector3(-3.0f,-4.0f,0.0f), Quaternion.identity);

//Audio
AudioSource audio;

audio=GetComponent<AudioSource>();
audio.Play(0);

//Tag
gameObject.tag;

//Active
GameObject obj = GameObject.FindWithTag("coin");
obj.SetActive(true/false);

//Finding objects
GameObject obj = GameObject.FindWithTag("tag");
GameObject obj = GameObject.Find("name");

//Finding components
GetComponent<Animator>(); //defaults to current object
obj.GetComponent<AudioSource>();

//Animation
Animator aninm;
void Start() { anim=GetComponent<Animator>(); }
void Update() {
    int state = anim.GetInteger("state");
    if (Input.GetKey(KeyCode.W))
        anim.SetTrigger("jump");
    if(Input.GetKey(KeyCode.D))
        transform.Translate(Time.deltaTime*2.0f,0.0f,0.0f);
        anim.SetInteger("state",0);
}
```

```
//Scenes
SceneManager.LoadScene(levelNames[currentLevel],LoadSceneMode.Additive); //LoadSceneMode.Additive means
load scene without unloading scene... idk why
SceneManager.UnloadSceneAsync(oldScene);

//MoveBackground
float speed = 0.1f;
float xpos = 0.0f;

void Start(){InvokeRepeating ("CreateAnchor", 0, 2.0f);}

  void Update() {
    xpos += Time.DeltaTime*speed;
    if(xpos >= 1.0f)
      xpos = 0.0f;
    Rendered bg = GetComponent<Renderer>();
    bg.material.mainTextureOffset = new Vector2(xpos,0.0f);

    float height = Camera.main.orthographicSize*2.0f;
    float width = Cmaera.main.aspect*height;
    float xChange = bg.bounds.size.x*speed*Time.DeltaTime;
    float xmin = -width/2.0f;

    GameObject[] anchors = GameObject.GetWithTag("anchor");
    foreach(GameObject anchor in anchors) {
      anchor.transform.Translate(-xChange,0.0f,0.0f);
      Renderer rn = anchor.GetComponent<Renderer>();
      if(anchor.transform.position.x < xmin - rn.bounds.size.x)
        Destroy(anchor);
    }

    void CreateAnchor() {
      float height = Camera.main.orthographicSize*2.0f;
      float width = Camera.main.aspect*height;
      float xmin = -width/2.0f;
      float ymin = -height/2.0f;
      float xmax = width/2.0f;
      float ymax = height/2.0f;

      GameObject anchor = Instantiate(anchorPrefab);
      Renderer r = anchor.GetComponent<Renderer>();
      float x = xmax + r.bounds.size.x;
      float y = Random.Range(ymin, ymax);
      anchor.transform.position = new Vector3(x,y,0.0f);
    }
  }
```