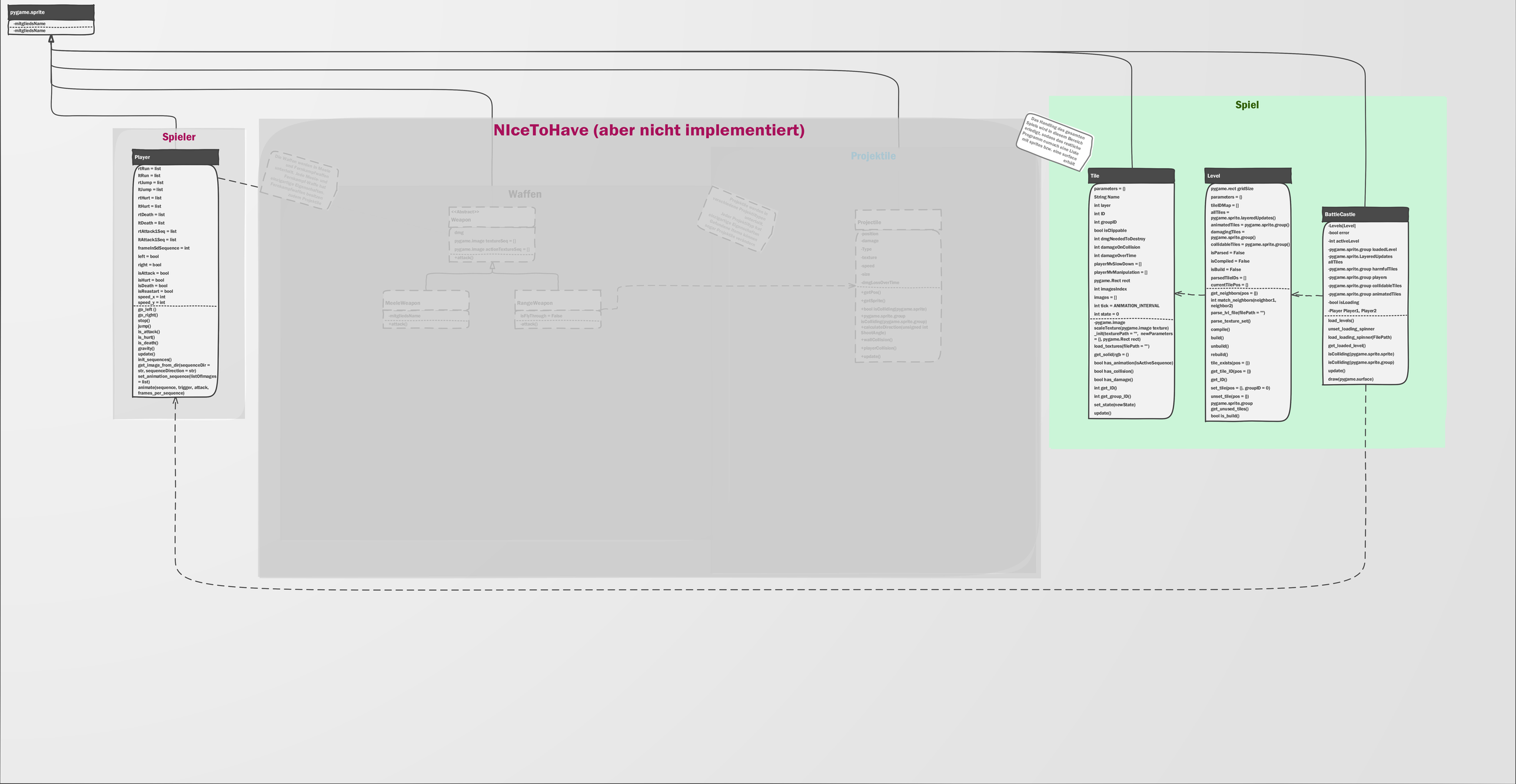
pygame.sprite
-mitgliedsName
-mitgliedsName
Spieler
Player
rtRun = list
ltRun = list
rtJump = list
ltJump = list
rtHurt = list
ltHurt = list
rtDeath = list
ltDeath = list
rtAttack1Seq = list
ltAttack1Seq = list
frameInSdSequence = int
left = bool
right = bool
isAttack = bool
isHurt = bool
isDeath = bool
isReastart = bool
speed_x = int
speed_y = int
go_left ()
go_right()
stop()
jump()
is_attack()
is_hurt()
is_death()
gravity()
update()
init_sequences()
get_image_from_dir(sequenceDir = str, sequenceDirection = str)
set_animation_sequence(listOfImages = list)
animate(sequence, trigger, attack, frames_per_sequence)
NIceToHave (aber nicht implementiert)
Die Waffen werden in Melee- und Fernkampfwaffen unterteilt. Jede Waffe hat einzigartige Eigenschaften. Fernkampfwaffen besitzen zudem Projektile.
Waffen
<<Abstract>>
Weapon
-dmg
-pygame.image textureSeq = []
-pygame.image actionTextureSeq = []
+attack()
MeeleWeapon
-mitgliedsName
+attack()
RangeWeapon
-isFlyThrough = False
-attack()
Projektile
Projektile werden in verschiedene Projektilarten unterteilt. Jeder Projektiltyp hat einzigartige Eigenschaften. Gelbschüsse können sogar Projektile vernichten.
Projektile
-position
-damage
-Type
-texture
-speed
-size
-dmgLossOverTime
+getPos()
+getSprite()
+bool isColliding(pygame.sprite)
+pygame.sprite.group isColliding(pygame.sprite.group)
+calculateDirection(unsigned int ShootAngle)
+wallCollision()
+playerCollision()
+update()
Das Handling des gesamten Spiels wird in diesem Bereich erledigt, sodass das restliche Programm nomoch eine Liste mit sprites bzw. eine surface erhält
Spiel
Tile
parameters = {}
String Name
int layer
int ID
int groupID
bool isClippable
int dmgNeededToDestroy
int damageOnCollision
int damageOverTime
playerMvSlowDown = []
playerMvManipulation = []
pygame.Rect rect
int imagesIndex
images = []
int tick = ANIMATION_INTERVAL
int state = 0
-pygame.image scaleTexture(pygame.image texture)
_init(texturePath = "", newParameters = {}, pygame.Rect rect)
load_textures(filePath = "")
get_solid(rgb = ())
bool has_animation(isActiveSequence)
bool has_collision()
bool has_damage()
int get_ID()
int get_group_ID()
set_state(newState)
update()
Level
pygame.rect gridSize
parameters = {}
tileIDMap = []
allTiles = pygame.sprite.layeredUpdates()
animatedTiles = pygame.sprite.group()
damagingTiles = pygame.sprite.group()
collidableTiles = pygame.sprite.group()
isParsed = False
isCompiled = False
isBuild = False
parsedTileIDs = []
currentTilePos = []
get_neighbors(pos = ())
int match_neighbors(neighbor1, neighbor2)
parse_lvl_file(filePath = "")
parse_texture_set()
compile()
build()
unbuild()
rebuild()
tile_exists(pos = ())
get_tile_ID(pos = ())
get_ID()
set_tile(pos = (), groupID = 0)
unset_tile(pos = ())
pygame.sprite.group get_unused_tiles()
bool is_build()
BattleCastle
-Levels[Level]
-bool error
-int activeLevel
-pygame.sprite.group loadedLevel
-pygame.sprite.LayeredUpdates allTiles
-pygame.sprite.group harmfulTiles
-pygame.sprite.group players
-pygame.sprite.group collidableTiles
-pygame.sprite.group animatedTiles
-bool isLoading
-Player Player1, Player2
load_levels()
unset_loading_spinner
load_loading_spinner(FilePath)
get_loaded_level()
isColliding(pygame.sprite.sprite)
isColliding(pygame.sprite.group)
update()
draw(pygame.surface)