TDDE25 ctf usermanual sg1-09 Joar, Felix, Svante
Implementerade features:
Hur man startar spelet?
Se till att du har pygame och pymunk v.5.7.0 nedladdade till din dator. Navigera dig till ctf-master katalogen för att sedan köra ctf.py i terminalen genom att skriva "python3 ctf.py". Om du använder VScode kan du istället köra ctf.py filen direkt från VScode.

Sound
Ljudet laddas in genom en separat fil sounds.py där varje ljud tilldelas en variabel så de går att komma åt från de andra filerna.
Ljudet som spelas då flaggan tas upp av en tank ligger i funktionen try_grab_flag, och ljudet spelas bara om tanken inte har den sänkta hastigheten som den senare får om den har flaggan, på så sätt spelas ljudet bara en gång eftersom tanken senare får den långsammare hastigheten.
Ljudet av att en tank skjuter spelas helt enkelt i shoot-funktionen.
När en trälåda går sönder spelas också ett ljud, detta görs i collision handlern för bullet och box.
Bakgrundsmusiken spelas inuti main-loopen.
Ljudet av att en tank exploderar spelas i collision handlern för bullet och tank, men bara när tanken har nått 0 hp.
När en tank vinner spelas ett ljud under has_won inuti main-loopen.

Hot seat multiplayer
För att kontrollera gamemode finns två booleans/flaggor: singleplayer och multiplayer. Dessa ges ett värde i startmenyn och avgör hur många tanks som görs till AI och hur många spelare som läggs till i player_list. Det går också att välja gamemode via terminalen där man skriver in ”Python ctf.py --singleplayer” eller ”Python ctf.py --hot-multiplayer”.

Hit points
Varje tank har ett attribut hit_points som standard är 2 men som sänks med 1 om en tank träffas av en bullet, och om den når 0 respawnar tanken. Detta sker i collision för bullet och tank.

Respawn protection
Om en tank når 0 hp sparas ticken när det hände i attributet respawn_time, en tank kan därefter inte tappa hp om det inte har gått 2500 ticks efter dess respawn_time.

Welcome screen / meny
När du startar spelet möts du av en meny som användaren kan navigera sig igenom med musen. I menyn kan du ändra karta och om du vill spela singleplayer eller multiplayer. Filen ctf.py importerar filen button.py som skapar, ritar och hanterar knappar. Den kan även rita text på skärmen. Filen består av en klass Button vars __init__ funktion skapar knappen efter givna parametrar. click metoden hanterar om musen är inom knappens rektangel och ändrar då färg. Den returnar sedan True om du har klickat på knappen. Sedan har vi metoden draw som ritar ut knappen med given text centrerad i rektangeln. Till sist har vi metoden write_text som bara är tillför att skriva ut text på skärmen.

Explosions (animerade)
När bullets kolliderar med något objekt så kommer en explosion inträffa i form av en animation, som hanteras i filen images.py. Funktionen läser in bildfilen, samt rader och kolumner som ger oss riktningen på hur animationen ska förekomma, och till sist x/y-värden som skalar ner bilden efter angivna värden.

A star search
En snabbare sökalgoritm än BFS. Istället för att utforska alla noder samtidigt utan diskriminering så ges varje nod två värden/poäng: vad det kostar att komma till noden från begynnelsenoden genom den nuvarande kortaste kända vägen, och hur långt det är till målet från noden. Dessa värden slås ihop till ett värde som blir den totala poängen av en nod. När grafen utforskas prioriterar

algoritmen alltid att först utforska de noder med låg poäng, på detta sätt har algoritmen en idé om riktning och kostnad av varje väg, och kan på så sätt ignorera vägar som inte är värda att utforska. En väg som t.ex. leder i motsatt riktning av målet skulle väldigt sent, om inte alls, utforskas.

Funktionalitet av varje fil:

Hur kallar den på button.py och Explosions i gameobjects.py?

För att skapa bakgrunden av gräs itererar programmet över kartan från maps.py och blittar bilder från data på skärmen, bilderna i sig laddas in i filen images.py.

Mycket av de saker som skapas i själva världen är objekt av klasser och deras relaterade funktioner från gameobjects-filen. De första fysiska objekten som skapas är alla boxes, vilka är instanser av en klass Box, som i sin tur ärver från klassen GamePhysicsObject vilket är en mer generell klass som hanterar saker med fysisk form och rörelse. Boxes skapas genom att funktionen create_boxes itererar över alla koordinater i den nuvarande kartan, som kommer från filen maps.py. Den skapar då boxobjekt och ger dem en viss bild från data och collision type bestämt av vilket värde de har i grafen för kartan. Detta görs med hjälpfunktionen get_box_with_type. De läggs sedan till i listan för game objects och space som hanterar dess fysik.

Flaggan skapas på ett liknande sätt men den är inte ett fysiskt objekt så den ärver inte av GamePhysicsObject och läggs inte till i space. Startkoordinaten av flaggan tas också från maps.py.
För att lägga till baserna kallas gameobjects.py igen för att skapa GameVisibleObject-instanser för att sedan lägga till dem i game_objects_list.

För att begränsa spelplanen där det inte finns boxes så läggs det till barriers. Dessa är segment som är static bodies, så att de blir som orörliga väggar kring planen. De läggs bara till i space eftersom vi inte är intresserade av att se dem.

Tanks skapas på ett liknande sätt som boxes. Funktionen create_tanks itererar över kartan och skapar objekt av Tank-klassen med motsvarande sprite från data-katalogen. De läggs både till i en separat lista bara för tanks för att underlätta manipulationen av dem, och vanliga game_objects_list. För att göra tanks till ai så går funktionen genom tanks_list och kallar på Ai-klassen från den separata ai.py-filen för att göra tankobjekt till Ai-objekt. En tank skjuter genom att kalla på funtkionen shoot som är definerad inuti klassen för tanks. Den skapar ett objekt av Bullet-klassen och lägger till den till listan och space på ett liknande sätt som de andra fysiska objekten.

Movement handlern kallar också på funktioner definierade i Tank-klassen för att få tillgång till deras rörelse och position för att spelaren ska kunna styra dem.

Hur objekt ska bete sig när de kolliderar med varandra bestäms i några collision handlers, där varje är specifik för att hantera en typ av kollision. Den enda som aktivt kallar på en annan fil är den för bullets och tanks, som använder många av metoderna för tanks, som att kolla hur mycket hp den har, hur länge sen den respawna och metoden för att respawna den.

Efter att alla objekt och assets är skapade så startar programmet main/game-loopen, där själva spelet tar plats. För varje iteration av spelet så kallas funktionen decide från ai-filen. Den används för att kontrollera hur ai:n beter sig, och bestämmer i stort sett hur den ska röra sig och om den ska skjuta. Inuti main-loopen kallas också många funktioner från Tank-klassen från gameobjects.py för att hantera allt som kan ske med en tank. De två huvudfunktionerna för varje tank är has_won och try_grab_flag som kollar om kraven för att någon att de två har uppfyllts, båda dessa är metoder av Tank-klassen.