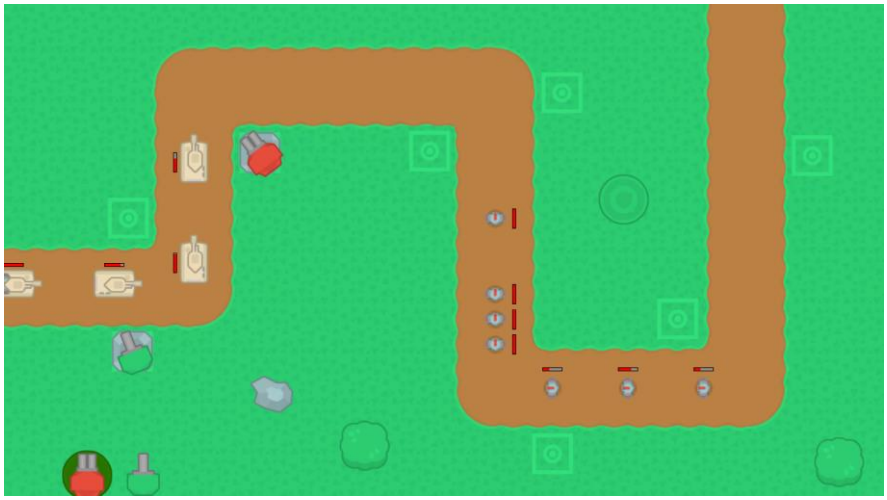
Unity Tower Defense

Af: Michael Hansen, Coding Pirates Furesø, 2022, version 0.90

Dokument og kode ligger her: https://github.com/mhfalken/unity/



Dette er en guide i hvordan man laver et simpelt Tower Defense spil i Unity, som vist på billedet. Det forudsætter lidt kendskab til Unity, som svarer til at man har lavet 2D Platform spillet, som er beskrevet i unity-intro.pdf. Det er også meningen at man skal bruge det omtalte dokument til opslag når der er noget man ikke kan huske eller finde ud af.
Spillet er bevist lavet simpelt så det er lettere at lave, dvs. der er steder hvor koden kunne være lavet mere optimal, men dermed også mere kompleks.

Dokumentet og koden er lavet i Unity version 2021.3. Den burde også virker i andre versioner, men der kan være små forskelle.

1 Guide

Denne guide er bevist lavet lidt overordnet, da det er meningen at man skal lave mere selv og dermed komme op på et højere Unity niveau.

1.1 Import start pakke

Opret et 2D game og importer https://github.com/mhfalken/unity/blob/main/tower_defense.unitypackage (**Assets->Import Package**).
Under *Graphics* ligger en masse grafik billeder som man kan bruge af. Hvis man mangler nogle, så kan man selv finde flere på nettet.
Under *Scripts* ligger nogle scripts som skal bruges undervejs i guiden.
Start med at vælge **Level1** under **Scenes** og slet *SampleScene*.

1.2 Kort og fjender

1. Start med at tegne en bane med en vej igennem under *Grid/Terrain* (brug **Tile Paletten**.)
2. Tag et billede af en fjende og sæt det ved start af vejen (uden for billedet). Kald det noget 'fornuftigt'.
3. Sæt **Order in Layer** til 1
4. Tilføj en **Collider 2d** og sæt **Trigger**. Sørg for at collideren er lidt større end figuren, da skuddene senere ellers har svært ved at ramme.
5. Tilføj **Rigidbody 2D** og sæt **Gravity Scale** til 0 og **Collision Detection** til **Continuous**.
6. Tilføj et **Tag** som hedder *Enemy*.
7. Tilføj scriptet *Enemy* til objektet.
8. Kør spillet og se hvad der sker.
9. Åben *Enemy* scriptet.

Det er *Enemy* scriptet som bevæger figuren. I `Update()` er der en `if` sætnings som gør at *fjenden* skifter retning for at følge vejen. Denne sætning skal rettes så *fjenden* følger den vej I har tegnet. For hvert sving skal der bruges en `if` sætning, så *fjenden* følger hele vejen. Her er en lille stump af koden fra *Enemy* script filen.

```
        if ((transform.position.x > -20) && (path == 0))
        {
            transform.rotation = Quaternion.Euler(0, 0, 90);
            path++;
        }
        else if ((transform.position.y > 11.5f) && (path == 1))
        {
            transform.rotation = Quaternion.Euler(0, 0, 0);
            path++;
        }
```

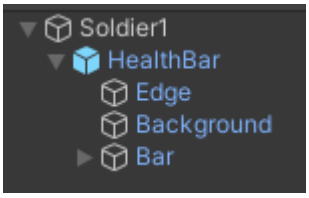
Man skal primært rette de ting som står med **rødt**. x/y angiver den retning figuren bevæger sig INDEN den skal dreje og >/< tegnet skal bruges alt efter hvilken retning den bevæger sig i. Tallet angiver hvilken position som den skal dreje på. Sidste tal i `Euler(0, 0, grader)` er drejningen af figuren i grader.
Ret nu koden så fjenden følger jeres vej. Det kræver lidt forsøg at så det til at passe. For hvert nyt sving skal man bruge en ekstra `if` sætning.
Når man er færdig kan det være en fordel at lave en **PreFab** af fjenden.

1.3 Health bar

For at vi kan skyde nogle forskellige fjender, har vi brug for nogle parametre til at styre hvor kraftige skud vi bruger og hvor meget et skud skader vores fjender før de dør.

Egenskab	Forklaring	Variabel navn
Hit impact	Hvor kraftig kuglen er. Dette tal starter ved 100.	hitImpact
Hit divider	Hvor meget fjenden kan klare. Dette tal starter ved 10.	hitDiv
Speed	Hvor hurtigt fjenden bevæger sig.	speed
Health	Hvor meget liv fjenden har tilbage. Det er et tal i intervallet [0-100]	health

For hvert skud som rammer udregnes skaden med følgende formel:
health = health - hitImpact/hitDivider
Når health er nul eller negativ, så dør fjenden.
Vi skal nu lave en health bar så man kan se hvor meget liv vores fjende har tilbage. Selve health bar'en er lavet som en **Prefab**, så den skal 'bare' tilsluttes til vores fjende.



Åben fjende **PreFab**'en (dobbelt klik på den) og tilføj *HealthBar* **PreFab**'en til den så det ser sådan ud (min fjende hedder Soldier1):

Tryk på *Soldier1* og træk *Bar* objektet over i feltet **Bar Obj**. Flyt health bar'en så den står et godt sted fx ovenover figuren. Gå ud af **PreFab** mode.
Selve koden til at styre health bar'en er allerede lavet i *Enemy* scriptet og ligger i funktionen `Hit()`.
Kør spillet og se at alt stadig virker.
Placer en lille række af fjender ud for start af vejen, så har man lidt at teste med.

1.4 Kanonkugle

Vi skal nu lav en kugle, som vores kanon kan skyde med.

1. Tag billedet af kuglen (*Bullet*) og træk det ind i **Scenen**.
2. Sæt **Order in Layer** til 10
3. Tilføj en **Collider 2D,** lav den lidt større end kuglen og sæt **Trigger**.
4. Tilføj scriptet *BulletCtrl.*

Sæt kuglen et 'godt sted' og se at den virker, når den rammer en fjende (health bar skal tælle ned og kuglen skal forsvinde).
Lav en **PreFab** når det virker.

1.5 Tårne

Vi skal nu lave et forsvarstårn og få det til at virke.

1. Opret et tårn med en kanon. (Det skal helst være to billeder, et af basen (Tower1, **Order in Layer** = 4) og så kanon som et under objekt (*Cannon*, **Order in Layer** = 9))
2. Tilføj en **Collider 2D** til *Cannon* som skal være så stor som så langt kanonen kan skyde og sæt også **Trigger**.
3. Tilføj *TowerCannon* scriptet til *Cannon*.
4. Tilføj *Bullet* fra **PreFabs** til **Bullet Obj** (*Cannon* **Inspector**)

Se om det virker, ved at få en fjende til at bevæge sig inden for rækkevidde af tårnet.
Tårnet kan skyde, men kanonen drejer ikke. Tilføj følgende linjer til *TowerCannon* scriptet:

```
float angle;

angle = AngleCalc(Vector3.up, dir);
transform.rotation = Quaternion.Euler(0, 0, angle);
```

Se at kanonen drejer inden den skyder.
Juster skudkraft og skudhastighed i **Inspector**'en og lav en **PreFab** af tårnet.
Her vil det være fint at lave nogle flere forskellige fjender og lave **PreFabs** af dem.

1.6 Platforme

Lige nu skal tårnene placeres inden spillet starter, hvilket jo ikke er meningen. Vi skal derfor lave nogle platforme som tårnene kan stå på.

1. Find et billede til platformen og træk det ind i **Scenen**. Kald det Platform.
2. Sæt **Order in Layer** til 3
3. Ret **transform->position z** værdien til -1
4. Tilføj en **Collider 2D**
5. Tilføj *PlatformCtrl* scriptet
6. I **Inspector**'en tilføj et element i **Towers Obj** og træk *Tower1* **PreFab**'en over i det.

Start spillet og se at man kan placere et tårn på platformen ved at trykke på den. (Hust at trykke i **Game** view)
Man kan desværre placere flere tårne på samme platform. Ret koden så der kun kan stå et tårn på hver platform.
Hint: Lav et `GameObject currentTowerObj = null;` som gemmer en peger til det tårn som står på platformen. `Instantiate` funktionen returnere en sådan peger.
Lav en **PreFab** og indsæt nogle flere platforme i spillet.

1.7 Game Master og tårn valg

Udfordringen med spillet er at der er behov for at holde styr på nogle resurser globalt set. Til det formål skal der oprettes en gamecontroller.

1. Opret et tomt objekt (**Create Empty**) og kald det *GameMaster*
2. Tilføj *GameMaster* scriptet

GameMaster scriptet skal bruges til at gemme globale variable, som kan tilgås fra alle de andre scripts. Lige nu indeholder det kun `towerSelected`.

Vi skal nu lave så man kan vælge mellem flere forskellige tårne.

1. Lav nogle flere forskellige tårne, så der er noget at vælge imellem
2. Tilføje dem til listen i *Platform* objektet (**Towers Obj**). (Husk at bruge **PreFab**'en!!)

Vi skal nu lave tårn vælgeren:

1. Find et billede af tårnet 1 og træk det ind i **Scenen**. Kald det *Tower1Sel*.
2. Set **Order in Layer** til 2
3. Tilføj en **Collider 2D**
4. Tilføj *TowerSel* scriptet
5. Ret **Tower Index** så det passer med listen i *Platform*. Første element i listen har indeks 0.

Gør det hele en gang til, men med et anden tårn billede og ret **Tower Index** så det passer.
Placer begge billeder ude i kanten af skærmen, så man kan bruge dem til at vælge hvilket tårn man vil indsætte.
Når man trykket på et af billederne, så bliver den globale variabel `towerSelected` sat til det valgte tårn.
Vi skal nu bruge den information til at indsætte det rigtige tårn på platformen, i stedet for bare at tage tårn 0.
Tilføj følgende linjer til *PlatformCtrl* scriptet.

```
GameMaster gameMasterObj;

gameMasterObj = GameObject.Find("GameMaster").GetComponent<GameMaster>();
```

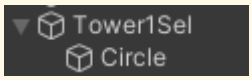
Ret nu koden så den bruger variablen `gameMasterObj.towerSelected` i stedet for `0` til at vælge hvilket tårn man sætter ind.
Test at koden virker, ved at sætte forskellige tårne ind på forskellige platforme.

Når man har valgt et tårn at sætte ind, kan man ikke se hvilket tårn man har valgt. Tilføj nu noget kode, som viser hvilket tårn som man har valgt til scriptet *TowerSel*.
Hint: Man kan lave et billede (firkant/cirkel) som tårnet står på, og så få dette billede til at være synligt når tårnet er valgt. Til det formål kan følgende linjer være nyttige:

```
GameObject circleObj;

circleObj = gameObject.transform.Find("Circle").gameObject;

if (gameMasterObj.towerSelected == towerIndex)
    circleObj.SetActive(true);
else
    circleObj.SetActive(false);
```



Her hedder mit bilede objekt *Circle*.
Test at selectoren virker.

1.8 Fjender spawner

Vi har nu brug for at der kommer fjender hele tiden. Til det formål skal vi lave et objekt som genererer fjender.

1. Find et godt billedet til en spawner (det vil kun kunne ses i **Scene** view ikke i selve spillet.)
2. Placer det ved start af vejen, uden for skærmen.
3. Tilføj scriptet *EnemySpawnerCtrl*
4. Tilføj alle fjenderne fra **PreFabs** til listen i **Inspector**'en (**Enemies Obj**).

Start spillet og se at der kommer fjender.
Det er generelt ikke nemt at lave en generator som laver fjender på en fed måde. Ret i koden, så der kommer flere forskellige typer fjender frem. Man kan eventuelt bruge følgende funktion, som laver et tilfældigt tal:

```
tal = Random.Range(MIN, MAX);
```

Hvor `tal` ligger fra og med MIN, til MAX, men uden MAX er med.
Matematisk skrives det således: `[MIN, MAX[`.
Eksempel: `Random.Range(4, 8)` vil lave et af følgende tal {4, 5, 6, 7}.
Det vil kræve nogle forsøg at få fjenderne til at komme på en rimelig måde.

1.9 Game Over

Hvis bare en fjende slipper igennem, så har man tabt.
Ændre koden i *Enemy* scriptet, så det skriver en tekst på skærmen, hvis en fjende slipper igennem. Det er lidt anderledes end 'normalt', da fjenderne bliver skabt on-the-fly og derfor ikke nemt kan linkes til teksten.

1. Opret en tekst som normalt og kald det *GameOver* og få det til at se godt ud (**UI->Legacy->Text**).
2. Husk i *Canvas* at sætte **Scale With Screen Size**.
3. For *GameOver* objekter, fjern fluebenet i **Inspector**'en for følgende felt (så teksten forsvinder):

4. I scriptet *Enemy*, indsæt følgende linje der hvor man har tabt spillet:

```
GameObject.Find("GameOver").GetComponent<Text>().enabled = true;
```

Det der sker er, at når man har tabt spillet, så bliver Game Over teksten synlig.
Prøv nu spillet og se om alt virker.

1.10Timer

TBD

2 Penge og gameplay

2.1 Penge

For at få et mere realistisk spil, skal der være nogle penge som man bruger på at købe tårne. Pungen til pengene skal være i vores *GameMaster*.

Tilføj følgende linjer til *GameMaster* scriptet:

```
[System.NonSerialized] public int towerCost = 0;
[System.NonSerialized] public int money = 200;
```

De 200 er vores startkapital.

Vi skal nu lave en tekst til at vise hvor mange penge vi har:

1. Tilføj et tekstfelt og kald det *Money*.
2. Formater feltet så det ser godt ud
3. Få *GameMaster* scriptet til at vise hvor mange penge vi har

Når man dræber en fjende, skal man have nogle penge. Tilføj denne linje til *Enemy* scriptet til at holde den værdi man skal have når en fjende dør (værdien kan ændres i **Inspector**en for hver fjende type):

```
[SerializeField] int money = 10;
```

Denne linje giver adgang til vores pengepung.

```
GameObject.Find("GameMaster").GetComponent<GameMaster>().money
```

Brug linjen i *Enemy* scriptet sammen med `money` feltet til at tælle vores penge op med, når en fjende dør.
Test spillet og se at vores penge tæller op hver gang en fjende dør.
Opdater alle fjender, så de giver forskellige penge at dræbe.

Det koster penge at købe et tårn. Det styres to forskellige steder.
I *TowerSel* scriptet skal man tilføje denne linje, som angiver hvad et tårn koster (sættes i **Inspectoren**):

```
[SerializeField] int cost;
```

Planen er at et tårn kun kan vælges når man har penge nok til at købe det. Ret *TowerSel* scriptet så billedet af tårnet kun er synligt når man har penge nok.
Hint: Brug disse linjer (hvis man har fulgt guiden med alle hints):

```
SpriteRenderer sr;

sr = GetComponent<SpriteRenderer>();

if (gameMasterObj.money >= cost)
    sr.enabled = true;
else
    sr.enabled = false;

if (gameMasterObj.towerSelected == towerIndex)
{
    if (sr.enabled)
        circleObj.SetActive(true);
    else
    {
        circleObj.SetActive(false);
        gameMasterObj.towerSelected = -1;
    }
}
else
{
    circleObj.SetActive(false);
}
```

Når man vælger tårnet skal man også gemme hvad prisen var. Tilføj følgende linje der hvor man vælger tårnet:

```
gameMasterObj.towerCost = cost;
```

Når man sætter et tårn ind, skal man tælle sine penge tilsvarende ned.
Ret scriptet *PlatformCtrl* i `OnMouseDown` til disse linjer:

```
if (currentTowerObj == null)
{
    if (gameMasterObj.towerSelected >= 0)
    {
        currentTowerObj = Instantiate(towersObj[gameMasterObj.towerSelected],
                                      transform.position, transform.rotation);
        gameMasterObj.money -= gameMasterObj.towerCost;
    }
}
```

Test spillet og se at alt virker. Det skal give penge at fjender dør, kun tårne man har råd til er synlige og når man indsætter et tårn, koster det penge.
Opdater så alle tårne koster noget forskelligt.

2.2 Fjender der skyder

For at spillet kan forsætte i noget tid, er det nødvendigt at fjenderne skyder og dermed ødelægger vores tårne.
Først skal vores tårne have en health bar.

1. Åben et tårn i **PreFabs** folderen
2. Tilføj health bar **PreFab**'en
3. Tilføj **Collider 2D** til basen
4. Tilføj **Rigidbody 2D** og sæt **Gravity Scale** til 0
5. Sæt tag *Tower*
6. Tilføj *Tower* scriptet
7. Sæt **Bar Obj** feltet i **Inspector**en

Lav ny kugle:

1. Lav en ny kugle (*BulletEnemy*)
2. Sæt **Order in Layer** til 10
3. Tilføj en **Collider 2D,** lav den lidt større end kuglen og sæt **Trigger**.
4. Lav et nyt script (*BulletEnemyCtrl*), hvor indholdet er kopiret fra *BulletCtrl* (Husk det KUN er indholdet af `class` koden som skal kopieres).
5. Ret `OnTriggerEnter2D` så det er *Tower* og ikke *Enemy* vi leder efter og rammer.

Sæt kuglen et 'godt sted' og se at den virker, når den rammer et tårn (health bar skal tælle ned og kuglen skal forsvinde).
Lav en **PreFab** af kuglen

Fjender skyder:

1. Åben en fjende i **PreFabs** folderen
2. Create Empty (*Gun*) under fjenden (soldat) [tank har allerede en Gun]
3. Tilføj en **Collider 2D** til *Gun* som skal være så stor som så langt den kan skyde og sæt også **Trigger**.
4. Tilføj *EnemyShoot* scriptet til *Gun*
5. Åben scriptet og fjern kommentar markeringerne (slet linje 45 og 63)
6. Træk *BulletEnemy* **PreFab** over i **Bullet Obj**.

Prøv spillet og se om alt virker. Fjenderne skal gerne skyde og ramme tårnene og når tårnene 'død' skal de forsvinde så man kan sætte nye tårne op.

2.3 Game play

Kunsten er nu at trimme alle parametrene så man får et godt gameplay, dvs. hvor meget de enkelte fjender skyder, hvor stor impact de har, hvor mange penge man får og bruger mv. Det skal alt sammen også passe til den fjende spawner funktion man har.

Her er mine tal til inspiration:

Fjende	Speed	Hit	Money	Impact	Rate	Range
Soldier1	10	10	20	0	0	0
Soldier2	10	12	50	100	1	4,5
Tank1	8	100	200	2000	0,3	8
Tank2	6	150	120	3000	0,2	7
Plane	15	75	150	4000	0,5	6
Tower1	-	100	100	500	3	9
Tower2	-	300	300	1500	1	8
Tower3	-	100	400	300	12	12