Webbutveckling I - Gesällprov

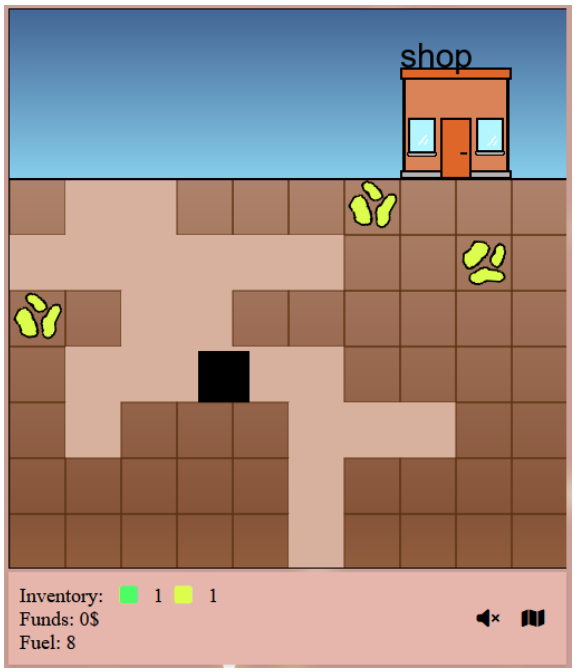
Amanda Lyvik

17 mars 2025

Stockholms Universitet
Institutionen för data- och systemvetenskap

Sammanfattning

Den här rapporten beskriver utvecklingen av en webbsida som innehåller ett interaktivt spel skapat med HTML, CSS och JavaScript. Projektet syftade till att bygga en spelupplevelse med dynamisk grafik, användarinteraktion och en responsiv layout. Rapporten redogör för hur kodbasen är strukturerad, hur spelets funktionalitet är uppbyggd, samt hur spelet ser ut och fungerar vid användning. Centrala delar av utvecklingen, som iterationer av spelkontrollen, presenteras av faktisk kod medans helheten, som spel-loopen, generering av spelvärld och resurser, samt interaktionen med affären presenteras i form av pseudokod. Dessutom visas visuella exempel på gränssnittet i olika stadier av spelet och på olika skärmstorlekar, för att illustrera användarupplevelsen och den dynamiska designen.



Figur 1: Spelvy under körning: spelaren har grävt ner för att hämta resurserna i marken.

Innehåll

1 Framtagande

I den här sektionen beskrivs hur koden har utvecklats från tanke till färdig produkt. Den fulla historiken över hur koden har utvecklats kan ses på GitHub [1].

1.1 Koncept

MotherLoad var ett gratis browserspel där spelaren kontrollerade en grävmaskin med målet att gräva sig ner i jorden, samla in resurser och återvända till ytan. Spelaren styr maskinen med piltangenterna för att navigera den tvådimensionella världen. På ytan finns en affär där resurserna kan säljas och spelaren kan köpa uppgraderingar. Det finns också en tankstation för bränsle.

Figur 2: Orginalspelet, MotherLoad.

Tanken bakom mitt projekt var att skapa en simplifierad version av detta spel. Då komplexiteten riskerar att växa snabbt med många funktionaliteter så fokuserade jag på att implementera de mest essentiella först:

- Generera en 2D värld med resurser
- Kontroll över en karaktär som kan plocka upp resurser
- Kamera som flyttas över världen så att spelaren alltid är i centrum

1.2 Tidig utveckling

För att uppnå de grundläggande mål som beskrivs i sektion 1.1 så skedde utvecklingen iterativt, vilket dokumenterades genom commits i versionshanteringen [1]. Nedan beskrivs de viktigaste förändringarna i de tidiga commit-stegen.

Commit 1 – Grundläggande struktur

`index.html`: En `<canvas>` lades till i HTML-filen för att skapa spelvyn.

`script.js`: För att kontrollera spelaren använder vi `eventListener` för `keydown` och `keyup`. Då flera tangenter kan hållas nere samtidigt möjliggörs diagonal förflyttning (kod i sektion 1.3.1). En enkel spel-loop implementerades också där spelarens position uppdaterades och canvasen ritas om under varje frame.

Commit 2 – Resursgenerering och kollision

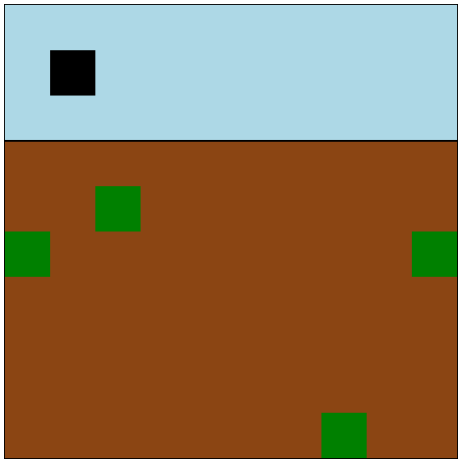
`script.js`: Funktionalitet för att generera resurser implementerades. `x` och `y`-koordinater slumpas fram och används för att generera ett resursobjekt som lagras i en array. En funktion för att detektera kollisioner, baserad på Axis-Aligned Bounding Box (AABB), lades till och används för att avgöra om spelaren är i kontakt med en resurs.

Commit 4 – Kamera och förbättrad rendering

För att kunna ha en värld som är större än spelfönstret så implementerades ett kameraobjekt. Kameran har positionsegenskaper och en funktion för att uppdatera dessa baserat på spelarens rörelse. Spelarens kontroller förbättrades så att spelaren inte kan gå utanför kartan (kod i sektion 1.3.2). Rendering av spelvärlden justerades baserat på kamerans position.

Commit 5 – Förbättrade spelarens rörelser och världens struktur

Ett ytskikt lades till på världen för att separera himmel och jord. Genereringen av resurser uppdaterades så de endast genereras under marknivå. En illustration av spelet kan ses i figur 3.



Figur 3: Spelläge vid commit 5

1.3 Iterationer av spelarkontrollen

Den mest komplicerade biten av kod är spelarkontrollerna. Hur spelaren rör sig är vad som påverkar allt annat i spelet och den måste hantera all interaktion med världen. Därför spenderades en del tid på att iterera spelkontrollerna efter att de mest essentiella delarna av spelet var på plats.

1.3.1 Enklaste lösningen

Som beskrivet i den förra sektionen så använder vi `eventListener` för `keydown` och `keyup` för att kunna hålla in fler knappar samtidigt för diagonal förflyttning.

```
let keys = {};
window.addEventListener("keydown", (e) => (keys[e.key] = true));
window.addEventListener("keyup", (e) => (keys[e.key] = false));
```

Den enklaste lösningen för att kontrollera spelarens position. `player.speed` är ett attribut i spelarobjektet som sätter antalet pixlar som spelaren förflyttas.

```
if (keys["ArrowUp"]) player.y -= player.speed;
if (keys["ArrowDown"]) player.y += player.speed;
if (keys["ArrowLeft"]) player.x -= player.speed;
if (keys["ArrowRight"]) player.x += player.speed;
```

1.3.2 Hålla spelaren på kartan

Nästa uppdatering var att använda kamerans position för att förhindra att spelaren rör sig utanför kartan.

```
  switch (true) {
    case keys["ArrowUp"]:
      if (player.y > camera.y) {
        // Prevent moving above the canvas top
        player.y -= player.speed;
      }
      break;
      ...
```

1.3.3 Rutnätsrörelse under jorden

I nästa iteration ska spelaren röra sig på olika sätt när vi är över och under jord. När vi gräver ska vi endast kunna röra oss i ett rutmönster. Diagonala steg är inte tillåtna. Vi behåller den tidigare koden för när vi rör oss över ytan, men måste lägga till en check när vi är under jord. En fördröjning ser till att varje steg tar en stund.

```
if (gridY > world.surfaceLevel) {
    if (currentTime - lastMoveTime >= moveDelay) {
      // Apply grid-based movement and restrict to one direction at a time
      if (keys["ArrowUp"] && !keys["ArrowLeft"] && !keys["ArrowRight"]) {
        player.y -= moveDistance; // Move up
    }
    ...
```

1.3.4 Typer per rutnätsruta

Rutor som vi har grävt igenom ska inte längre hanteras som mark. När vi genererar världen så lägger vi till ett attribut för rutnätet där vi sätter en typ för varje ruta. Nu kan vi genom att observera de närliggande rutornas typer se om de ska hanteras som mark (`ground`) eller luft (`air`). Jag ändrade också fördröjningen till en animering som hanteras av `startDigging` funktionen.

```
if (keys["ArrowUp"]) {
    if (neighbors.above && neighbors.above.type === "ground") {
      startDigging(neighbors.above);
    } else {
      player.y -= player.speed;
    }
    ...
```

1.3.5 Masscentrum

Ett problem som uppstod var att eftersom spelaren kan röra sig fritt över jorden avgörs vi vilken ruta spelaren är i baserat på dess masscentrum. Vi kan alltså trigga `startDigging` när spelarens masscentrum är i en ruta bredvid en markruta utan att vara i kontakt med den. Samma princip gjorde att spelaren kunde passera igenom markrutor.

```
  if (keys["ArrowUp"]) {
    if (neighbors.above?.type === "ground") {
      if (checkCollision(player, neighbors.above)) {
        console.log("collide: above");
      } else {
        let distanceToWall = player.y - (neighbors.above.y + WORLD.GRID_SIZE);
        player.y -= Math.min(player.speed, distanceToWall);
      }
    } else {
      let stopMovingUp = false;
      // Check if either aboveLeft or aboveRight is ground and block movement
          if necessary
      if (
        neighbors.aboveLeft?.type === "ground" &&
        checkCollision(player, neighbors.aboveLeft)
      ) {
        stopMovingUp = true;
      }
      if (
        neighbors.aboveRight?.type === "ground" &&
        checkCollision(player, neighbors.aboveRight)
      ) {
        stopMovingUp = true;
      }

      if (!stopMovingUp) {
        player.y -= player.speed;
      }
    }
  }
```

Utöver detta så påverkas spelaren av gravitationen och vi har fler checkar som gör att vi inte kan gräva uppåt och inte åt sidorna om vi inte står gå mark. Spelkontrollerna är inte perfekta men fungerar tillräckligt för att gå vidare med resten av spelet.

1.4 Affären

Det sista större tillägget i spelet var implementeringen av affären, som spelar en central roll för spelets vinstvillkor. Affären representeras av ett objekt samt ett antal rutor i världens rutnät med typen `shop`. Detta gör det enkelt att hantera kollision med affären i spelkontrollfunktionen.

```
1    if (keys["ArrowUp"]) {
2      if (
3        neighbors.above?.type === "ground" ||
4        (neighbors.above?.type === "shop" && world.grid[gridY][gridX] !== "shop")
5      ) {
6        if (checkCollision(player, neighbors.above)) {
7          console.log("collide: above");
8        } else {
9        ...
10   }
```

I spel loopen, efter att spelaren förflyttats, kontrollerar vi om spelaren befinner sig i affären vilket pausar spelet och öppnar affärens fönster. I affären finns tre knappar: `köp`, `sälj` och `stäng`. Genom att sälja resurser får du pengar, med pengarna kan du köpa bränsle. Säljer du alla resurser så vinner du spelet. Får du slut på bränsle så förlorar du.

1.5 Fyll ut sidan

När spelets essentiella funktioner var på plats så lades dessa detaljer till för att för att berika användarupplevelsen.

Starta spelet: När sidan laddas så är canvasen blank med en knapp för att starta spelet. När spelet tar slut så visas även en knapp för att starta om spelet. Se figur 5, 6c och 6d.

Ljudeffekter: Samtidigt som spelet startar så startar en ljudslinga. Toggle knappen 🔇 🔊 under canvasen pausar och startar musiken. När spelaren köper, säljer, gräver genom resurserna, vinner eller förlorar så spelas också en tillhörande ljudeffekt.

Lager display: Under spelet finns en display för spelarens lager, pengar och bränsle. Först visades endast antalet resurser som spelaren plockat upp. När fler typer av resurser lades till så uppdaterades displayen till att visa antal av varje typ. Se figur 6d.

Skaka kameran: När spelaren gräver sig in i en ruta som innehåller en resurs så skakar kameran.

Kartan: Bredvid toggle knappen som startar och stoppar musiken finns en kart-knapp 🗺. När den klickas så öppnas ett modalt fönster som visar upp världskartan. För att stänga kartan igen så klickar man på ✖ knappen uppe i högra hörnet. Se figur 7.

Bilder: Istället för de gröna rutorna som illustreras i figur 3 så används bilder för att rita upp resurserna och affären (figur 6a). En funktion för att byta färger på bilderna implementerades för att kunna använda samma figur för resurser med olika färger.

Sidan: Layouten är responsiv för att hantera olika stora skärmar. Bakgrunden har också interaktiva partiklar för att göra den mer levande. En `aside` sektion inkluderades för att beskriva spelreglerna och en `nav` för att länka till resurser som används. Se figur 5.

Fler typer av resurser: För att hantera fler typer av resurser gjordes ett flertal ändringar. Innan resurserna genereras så skapas en bild av resursen med rätt färg. Affären uppdateras också för att visa resursernas olika värden. Varje typ av resurs har ett eget värde och djup i värden där de förekommer.

2 Form

Den här sektionen beskriver hur sidans färdiga kod är uppbyggd och fungerar. Fokus ligger på HTML-strukturen, CSS-stilar och de JavaScript-moduler som driver spelet. Exempel på centrala kodavsnitt inkluderas för att illustrera viktiga delar av implementeringen.

2.1 HTML

Sidan är byggd med en enda HTML-fil, `index.html`. Huvudinnehållet har placerats i en wrapper `div` för att separera den från bakgrunden. I huvudinnehåll har vi en `header`, `nav`, `main` och `footer`. `main` innehåller spelet och en `aside` med instruktioner. Spelet ligger i `canvasContainr` tillsammans med `startButton`, `shopWindow` och `finishedGameWindow` så att de kan positioneras i mitten av canvasen (kod i sektion 2.2). I slutet ligger ljudfilerna som används av spelet.

```
<!DOCTYPE html>
<html lang="sv">
<head>
    <link href="styles.css" rel="stylesheet">
    <link href="layout.css" rel="stylesheet">
    <link href="theme.css" rel="stylesheet">
    <script type="module" src="main.js" defer></script>
    <script
      src="https://kit.fontawesome.com/3f6618ab3b.js"
      crossorigin="anonymous"
    ></script>
    <script type="text/javascript" src="particles.js"></script>
</head>
<body>
    <div id="particles"></div> <!-- Dynamisk bakgrund -->
    <div class="wrapper">
        <header>Sidans titel</header>
        <nav> <!-- Navigationsmeny --> </nav>
        <main>
            <section>
                <header>Spel titel</header>
                <div id="gameInfoContainer">
                 <div id="canvasContainer">...</div>
                <aside> <!-- Spel instruktioner --> </<aside>
                </div>
                <audio> <!-- Ljudeffect --> </audio>
            </section>
        </main>
        <footer> <!-- Kontakt information --> </footer>
    </div>
</body>
</html>
```

2.2 CSS

Som vi kan se i sektion 2.1 så använde tre stycken CSS filer. Stilmallen är uppdelad i flera filer för att hålla en tydlig separation mellan layout, teman och generella stilar.

- `layout.css` hanterar positionering av element med hjälp av `flexbox` (och `grid`). Media queries används för att anpassa layouten till olika skärmstorlekar.

```
@media (min-width: 800px) {
  #gameInfoContainer {
    flex-direction: row;
  }
  #canvasContainer {
    flex: 2;
  }
  aside {
    flex: 1;
  }
}
```

`translate` används för att med placera spelfönster i mitten av canvasen.

```
#finishedGameWindow {
  position: absolute;
  top: 50%;
  left: 50%;
  transform: translate(-50%, -50%);
}
```

- `styles.css` innehåller visuella detaljer såsom färger, filter, skuggor och kantlinjer.

```
aside {
  background-color: var(--game-controls);
  box-shadow: 0 2px 4px rgba(0, 0, 0, 0.1);
}
```

- Variabels `--game-controls`, som användes i exeplet ovan, definieras i `theme.css` filen. Jag definierar de variablerna så att dom kan användas dynamiskt i JavaScript.

```
:root {
  --game-controls: #e6b6ac;
}
```

2.3 Script

I sektion 2.1 ser vi att sidan använder tre script. Två har importerats från externa källor. Ett kit från `fontawesome` används för att inkludera ikoner. [2], samt `particles.js` för att hanterar de interaktiva partiklarna i bakgrunden [3].

Spelets funktionalitet hanteras huvudsakligen i `main.js`, som importerar och organiserar funktionalitet från flera moduler:

- `worldSetup.js` genererar och ritar upp spelvärlden.
- `player.js` hanterar spelarens rörelser och interaktioner.
- `shop.js` implementerar affärens funktionalitet.
- `config.js` innehåller konstanter för att kontrollera spelet.
- `background.js` och `particles.js` sköter den dynamiska bakgrunden.
- `help.js` tillhandahåller hjälpfunktioner för att läsa CSS-variablerna från `theme.css` och ändra färg på bilder.

2.4 Pseudokod för helheten

I den här sektionen finns pseudokod för olika delar av sidan. För att underlätta läsligheten har den delats upp i ett flertal algoritmer.

Algorithm 1: Översikt av sidans funktionalitet

onLoad
  Ladda `HTML`, `CSS` och `Script`
  Initiera dynamisk bakgrund
end
if *användaren klickar på Starta spel"* **then**
  ladda omfärgade bilder för resurserna
  göm startknappen
  starta bakgrundsmusik
  generera spelvärlden
  uppdatera affärens display
  starta spel-loopen
end
if *användaren klickar på navigationslänk* **then**
  öppna länk i ny flik
end

Algorithm 2: Översikt över start av spelet

vänta på att resursernas bilder genereras
göm startknappen
visa spelkontroll området
spela musiken
generera värdens grid
generera resurser
uppdatera affären med info om resurserna
starta spel loopen

Algorithm 3: Översikt över spel loopen

while *spelet är aktivt* **do**
  if *spelaren rör sig* **then**
    uppdatera spelarens position
    uppdatera bränslenivå
    uppdatera bakgrund baserat på djup
  end
  if *spelaren kolliderar med resurs* **then**
    flytta resurs till spelarens lager
  end
  if *spelaren går in i affären* **then**
    öppna affären
  end
  if *spelaren går nära kanten* **then**
    uppdatera kamerans position
  end
  rendera spelvärlden
  if *vinst eller förlust* **then**
    öppna "avslutat spel" fönstret
  end
end

Algorithm 4: Översikt över affären

```
while affären är öppen do
    if användaren klickar på "Buy" then
        uppdatera spelarens pengar
        uppdatera spelarens lager
        spela ljudeffekt
    end
    if användaren klickar på "Sell" then
        uppdatera spelarens pengar
        uppdatera spelarens bränsle
        spela ljudeffekt
    end
    if användaren klickar på "Cancel" then
        stäng affären
    end
end
```

Algorithm 5: Översikt över avslutat spel

```
while "avslutat spel" fönstret är öppet do
    spela ljudeffekt
    vänta på användarinput
    if användaren klickar på "Starta om spel" then
        starta om musiken
        återställ världen
        stäng "avslutat spel" fönstret
        starta spel loopen
    end
end
```

Algorithm 6: Översikt över genereringen av spelvärlden

```
// generera rutnätet
for varje rad ruta i världen do
    if rutan är i affären then
        sätt rutan typ till "shop"
    else if rutan är över ytan then
        sätt rutan typ till "air"
    else
        sätt rutan typ till "ground"
end
// generera resurserna
for varje typ av resurs do
    while inte placerat alla do
        generera slumpmässiga x och y på tillåtet djup
        if resurs inte finns på ruta (x, y) then
            lägg en resurs på ruta (x, y)
        end
    end
end
```

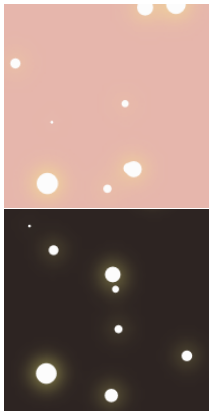
Algorithm 7: Översikt över renderingen

```
rensa canvasen
definiera gradienter för mark och himmel
rita rektanglar för mark och himmel
for varje rad ruta i världen do
    if rutan är synlig then
        if rutan är "ground" then
            rita en kant runt rutan
        else if rutan är "air" och under ytan then
            fyll rutan med ny färg
    end
end
rita affärens bilder
rita linje för markytan
for varje resurs do
    rita resursens bild med random rotation
end
rita "Shop" texten
rita spelaren
```

3 Funktion

Den här sektionen beskriver hur den färdiga sidan fungerar vid körning, med fokus på användarupplevelsen och visuella funktioner.

3.1 Bakgrunden



Figur 4: Dynamisk bakgrundsfärg

När sidan laddas initialiseras en partikelanimation i bakgrunden med hjälp av `Particles.js` [3]. Partiklarna rör sig långsamt över skärmen och reagerar på användarens muspekare genom att växa i storlek vid hovring. Antalet partiklar ökar också om användaren klickar med muspekaren.

Utöver partikeleffekterna uppdateras även bakgrundsfärgen dynamiskt under spelets gång. Färgen skiftar gradvis beroende på hur djupt spelkaraktären har grävt sig ner i världen. Vid ytan är bakgrunden ljusare, medan den blir mörkare ju längre ner spelaren tar sig. Denna effekt skapas genom att varje gång spelarens djup förändras anropas funktionen `updateBackground()`, som interpolerar mellan två färgtoner.

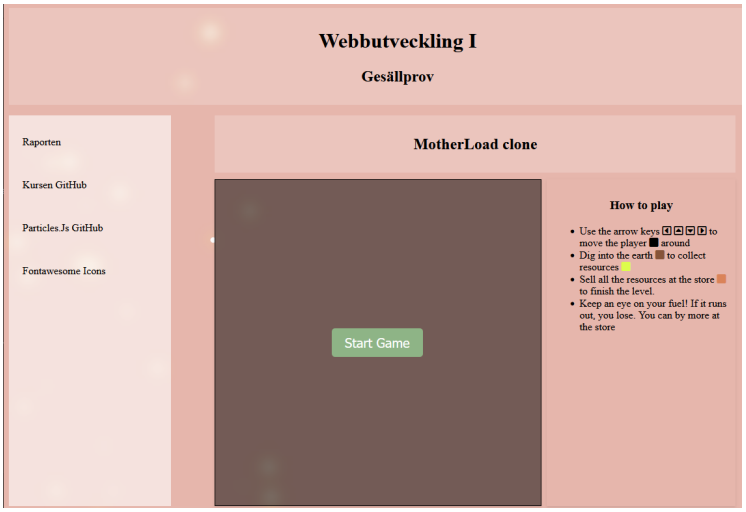
Figur 4 visar en jämförelse mellan bakgrunden nära ytan (övre bilden) och djupt ner under marken (nedre bilden).

3.2 Layout

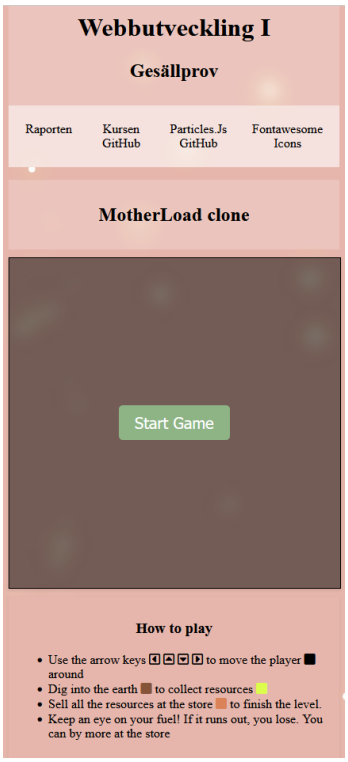
Layouten är responsiv och anpassar sig automatiskt efter användarens skärmstorlek. Figur 5 illustreras de två extrema tillstånden med den smalaste och bredaste layouten.

Layouten för smala skärmar visas i figur 5b till höger. I detta läge placeras navigeringsfältet ovanför canvasen, medan spelinstruktionerna läggs under spelet.

På bredare skärmar, som illustreras i figur 5a nedan, utnyttjas det tillgängliga utrymmet genom att placera både navigeringsfältet och instruktionerna vid sidan av canvasen.



(a) Vy på stor skärm



(b) Vy på liten skärm

Figur 5: Gränssnittets anpassning för olika skärmstorlekar

3.3 Spelet

Spelet har flera distinkta tillstånd som påverkar både spelmekaniken och gränssnittet. Figur 6 visar fyra centrala stadier i spelupplevelsen:

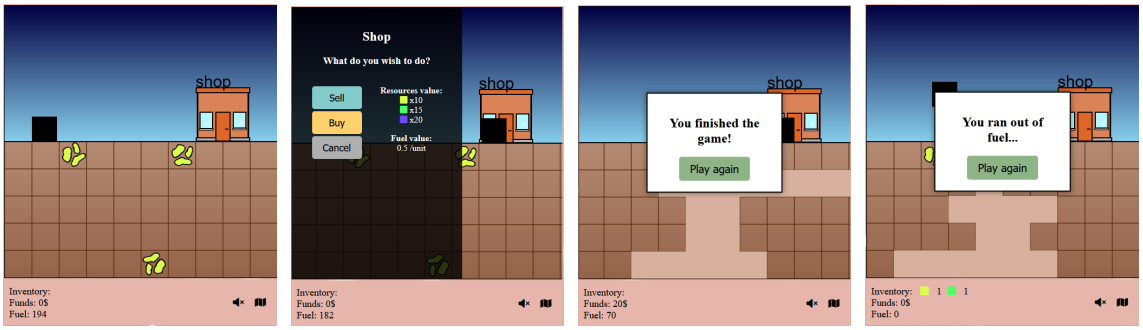
(a) **Startläget** visas direkt efter att användaren klickat på Start Game"knappen och spelet initierats. Spelaren kan nu börja gräva efter resurser och utforska världen genom att använda piltangenterna.

(b) **Affärsläget** aktiveras när spelaren går in i butiken. Här pausas spelvärlden, och spelaren kan sälja resurser samt köpa bränsle för att fortsätta gräva.

(c) **Vinsten** inträffar när spelaren har samlat in och sålt alla resurser. Ett meddelande visas som bekräftar att målet är uppnått, och spelet kan startas om. I den här figuren ser vi också hur tunnlarna ritas upp.

(d) **Förlustläget** visas när spelaren får slut på bränsle innan alla resurser har samlats in. Ett förlustmeddelande visas och spelaren får möjlighet att försöka igen.

Dessa tillstånd är avgörande för att strukturera spelets flöde och ge spelaren tydlig återkoppling om framsteg, status och nästa möjliga steg.



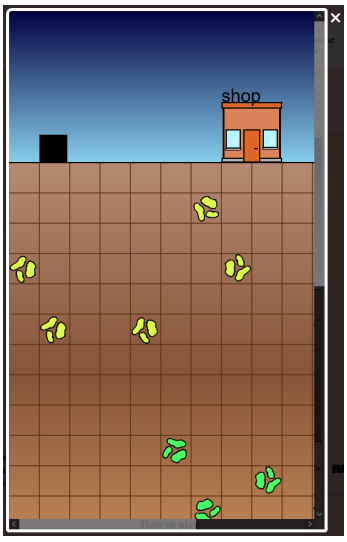
(a) **(b)** **(c)** **(d)**

Figur 6: Olika stadier i spelet: (a) start, (b) i affären, (c) efter vinst, (d) efter förlust

3.3.1 Kartan och världens utseende

Under spelvyn i figur 6 kan vi också se spelets nuvarande tillstånd med spelarens "Inventory", "Fuel" och "Funds" till vänster och två ikon knappar till höger. Knapparnas funktion beskrivs i sektion 1.5. I figur 7 till höger illustreras modal fönstret med kartan som öppnas när man klickar på kartknappen 🗺.

Notera att utseendet på världen är baserad på parametrarna i `config.js` och kan varieras för att bland annat ändra storleken på världen. I figur 6 är världen 10x10 rutor stor vilket gör att vi ser hela världen samtidigt i vår 10x10 canvas. I figur 7 är världen mycket större. Vi kan se hur gradienten i marken sträcks för att fylla ut den större världen. Om canvasen hålls i samma storlek som för den mindre världen så kommer botten inte att synas när spelet startar. Istället så kommer kameran scrolla ner i takt med att spelaren gräver.



Figur 7: Karta över spelvärden

Referenser

[1] Amanda Lyvik. URL: https://github.com/AmandaLyvik/Webbutveckling-I/compare/main...ges%C3%A4llprov.

[2] URL: https://fontawesome.com/.

[3] Vincent Garreau. URL: https://github.com/VincentGarreau/particles.js.git.