• *The* classic word game, called Alfapet in Swedish.
• Invented by Alfred Mosher Butts in 1938.
A simple enough game:

- 2–4 players, each with 7 letter tiles.
- Players take turns placing 1–7 tiles on the 15x15 board s.t.:
  - The tiles placed form a horizontal and/or vertical line.
  - The line is “dense”, no gaps.
  - All words (gap-free lines) formed on the board are real English words.
- Score the words formed (points summed on all words created, modified by bonus squares).
- Turn ends by drawing new tiles.
The problem

What is the problem?

1. What is my highest-scoring next move?
2. What is my best next move ignoring opponents?
3. What are my best next $k$ moves, given $n$ tiles ignoring opponents?
4. What is my best move considering the opponent?

This gets very difficult. What can we reasonably do?
• Problem 1 is simplest. This ignores the opponent, the future state of the board and the tiles left.
  • Hopefully these can be reintroduced to a successful solution?
• Bonus tiles and letter scores are also complicated, we can try generalising these away:

\[ f_{\text{score}}(\text{Board, Word, Placement}) \rightarrow \mathbb{N} \]

  • Assume that \( f_{\text{score}} \) is efficient (it is!)
  • But “forget” how it works to simplify the definition.

• Ignoring the future of the board makes its 2D-ness ignorable:

```
  *   *   H   A
  B   E   E
  N
  D

  *   N   *
  B   *   *
  H   E   *

  *   E   N   D
  B   E   E   *
  D
  A   *
```
What kind of *-constraints are there?

- Single letters and empty positions are trivial constraints.
- $c_1$ says “Any letter that forms a word when followed by the letter $b$” (no such letter exists in English).
- $c_2$ says “Any letter which forms a word when followed by the word end” (b, f, l, m, p, r, s, or t).
- Also, constraints saying “Any letter $\alpha$ such that $w$ and $w'$ form a word when concatenated as $w\alpha w'$.” are possible
  - $w$ and $w'$ are either (non-word) letters or words
  - For example, if $be$ and $ate$ are placed with a blank in between them the letter $r$ may be placed to form the word $berate$. 
The problem has three parts: \((B, T, D)\) where

- **B**, the board, is a set of constraint sequences
  - 30 sequences of length 15 in classic scrabble
- **T** is a multi-set of letters from \(\{a, \ldots, z\}\)
  - 7 letters in classic scrabble
- **D** is a set of allowed words
  - More than 600,000 words in the Oxford English dictionary

So, \(|D| \gg |B| + |T|\). A problem? Not really, \(D\) is constant.

Moves change \(B\) and \(T\) continuously, whereas the OED remains the same for decades at a time \(\implies\) we can do all the preprocessing we want on \(D\) (good!) but need to be careful with the complexity in \(D\) in practice (bad!)
Let us properly define the possible constraints. Let $\Sigma$ be the alphabet and $D$ the dictionary ($D \subset \Sigma^*$) in the following.

**Definition (Prefix constraint)**

If $w \in D \cup \Sigma$ and $\alpha \in \Sigma$ let $\alpha \in c_{\text{prefix}}(w, D)$ iff $\alpha w \in D$.

**Definition (Suffix constraint)**

If $w \in D \cup \Sigma$ and $\alpha \in \Sigma$ let $\alpha \in c_{\text{suffix}}(w, D)$ iff $w \alpha \in D$.

**Definition (Infix constraint)**

If $w \in D \cup \Sigma$ and $\alpha \in \Sigma$ let $\alpha \in c_{\text{infix}}(w, w', D)$ iff $w \alpha w' \in D$.

Not so interesting: $c_{\text{blank}} = \Sigma$, $c_{\text{letter}}(\alpha) = \alpha$. 
Precomputation!

Since \( D \) seldom changes we can just precompute the constraints!

- Construct a hash-map \( c_{\text{prefix}, D} \) which for all \( w \in D \) gives
  \[ c_{\text{prefix}, D}(w) = c_{\text{prefix}}(D, w). \]
- Same for suffix.
- Almost the same for infix. Construct
  \[ c_{\text{infix}, D}(w, w') = c_{\text{infix}}(D, w, w') \text{ iff } c_{\text{infix}}(D, w, w') \neq \emptyset \]
  Leave other \((w, w')\) unmapped.

- All pairs would give a map of size \(|D|^2\).
- Statistical experiments suggest that merely tens of thousands of “valid” pairs exist in English.
- Swedish and German might be worse?
Getting rid of the board

With dictionary precomputations done in advance taking the board

\[
\begin{array}{ccc}
 & H & A \\
B & E & E \\
N &   &   \\
D &   &   \\
\end{array}
\]

it is not hard to construct an efficient algorithm to translate each row and column into a sequence of letters and sets of letters:

\[
(\emptyset, \{b, f, l, m, r, s, t, v, w\}, h, a), \quad (\Sigma, b, \{a, i, o\}, \{a, i\}),
\]
\[
(b, e, e, \{d, h, m, n, s, t, x, y\}), \quad (\{a\}, e, n, d)),
\]
\[
(\{e, y\}, n, \{m, n, p, r, s, w, x, y\}, \Sigma), \quad (h, e, \{o, u\}, \{o\}),
\]
\[
(\Sigma, d, \Sigma, \Sigma), \quad (a, \{f, n, p, r, s, t\}, \Sigma, \Sigma)
\]
Finally, formalisation

With a fixed dictionary $D$, alphabet $\Sigma$, and scoring function $f_{\text{score}}$:

**Definition (Scrabble problem instance)**

A Scrabble problem instance is a tuple $(B, T)$ where $T$ is a finite multi-set over $\Sigma$ and $B$ is a finite subset of $(\Sigma \cup \mathcal{P}(\Sigma))^*$. 

**Definition (Placement)**

A valid placement for a Scrabble problem instance $(B, T)$ is a tuple $(\alpha_1 \cdots \alpha_n, \beta_1 \cdots \beta_m, i) \in D \times B \times \mathbb{N}$ such that

- $n + i < m$, and
- $\beta_{i-1}$ and $\beta_{n+i+1}$ are both sets if they exist, and
- $\alpha_j = \beta_{i+j}$ or $\alpha_j \in \beta_{i+j}$ for all $j \in \{1, \ldots, n\}$, and
- letting $P = \{\alpha_j \mid j \in \{1, \ldots, n\}, \beta_{i+j} \text{ is a set}\}$, $P \subseteq T$ and $P \neq \emptyset$. 
Definition (Scrabble problem solution)

For a Scrabble problem instance \((B, T)\) a placement \((w, b, i)\) is a solution if it maximises \(f_{\text{score}}\).

We need to enumerate all possible placements (otherwise we need to analyse \(f_{\text{score}}\)). There shouldn’t be *that* many (?)

1. Set \(v_{max} = 0\).
2. Take the next \(b = \beta_1 \cdots \beta_m \in B\).
3. Intersect each set \(\beta_j\) with \(T\).
4. If some set in \(b\) is empty, go to 2.
5. For each word \(w\) in \(D\) which matches \(b\): \(\text{<<DANGER>>}\).
   - Validate that \((w, b, i)\) is a placement for some \(i\).
   - Score \((w, b, i)\) using \(f_{\text{score}}\), set \(v\) to the score.
   - If \(v > v_{max}\) let \(v_{max} = v\).
6. Go to 2.
• Steps 1–4 and 6 are small stuff, $O(|T| \sum \{|b| \mid b \in B\})$.
• A naive implementation of step 5 is trouble. $|B||D|$ attempts to match words to constraints.
• For now, assume few words match. Quick way to find them?
My first approach: hash-map indexing letter-pairs.
Looking at this subproblem:

**Definition**

Given $b \in (\Sigma \cup \mathcal{P}(\Sigma))^*$ as input does there exist a word $w \in D$ such that $w$ matches $b$ in the sense of a placement?

Complexity in terms of $|D|$ still considered, but preprocessing of $D$ is allowed.

Construct a hash-map $h : \Sigma \times \mathbb{N} \times \Sigma \to \mathcal{P}(D)$, which, for $\alpha_1, \alpha_2 \in \Sigma$ and $i \in \mathbb{N}$ gives $W = h(\alpha_1, i, \alpha_2)$ where $W$ is all words which contain the letter $\alpha_1$ and $\alpha_2$ at positions $i$ steps apart. That is,

$$h(x, 5, l) = \{\text{anxiously, expertly, inexorably, maximally, obnoxiously, paradoxically, textually, textural}\}$$

Pretty large but not difficult pre-processing.

The ispell american-english dictionary of 98,569 yields a map with 2,996,606 words. A key maps to an average of 492.54 words.
Matching \((\Sigma, b, \{a, i, o\}, \{a, i\})\) can then be speeded up (?) by matching it to the words in

\[
(h(b, 0, a) \cup h(b, 0, i) \cup h(b, 0, o)) \cap (h(b, 1, a) \cup h(b, 1, i))
\]

instead of all of \(D\) (408 words in ispell american-english).

This depends on the structure of English words for speed. Hard to analyse. For other languages maybe \(h\) will map to gigantic classes?
If we avoid relying on the structure of $D$ then scanning may be necessary. How do we do scanning the best?

We can construct a deterministic finite automaton that attempts to match all of $B$ at once!