RL-KLM: Automating Keystroke-level Modeling with Reinforcement Learning

Katri Leino
Aalto University, Finland
katri.k.leino@aalto.fi

Antti Oulasvirta
Aalto University, Finland
antti.oulasvirta@aalto.fi

Mikko Kurimo
Aalto University, Finland
mikko.kurimo@aalto.fi

ABSTRACT

The Keystroke-Level Model (KLM) is a popular model for predicting users' task completion times with graphical user interfaces. KLM predicts task completion times as a linear function of elementary operators. However, the policy, or the assumed sequence of the operators that the user executes, needs to be prespecified by the analyst. This paper investigates Reinforcement Learning (RL) as an algorithmic method to obtain the policy automatically. We define the KLM as an Markov Decision Process, and show that when solved with RL methods, this approach yields user-like policies in simple but realistic interaction tasks. RL-KLM offers a quick way to obtain a global upper bound for user performance. It opens up new possibilities to use KLM in computational interaction. However, scalability and validity remain open issues.

CCS CONCEPTS

• **Human-centered computing** → *HCI design and evaluation methods*; **User interface design**; • **Computing methodologies** → *Machine learning*.

KEYWORDS

Keystroke-level modelling, Reinforcement learning, Computational evaluation, Computational design

ACM Reference Format:
Katri Leino, Antti Oulasvirta, and Mikko Kurimo. 2019. RL-KLM: Automating Keystroke-level Modeling with Reinforcement Learning. In *24th International Conference on Intelligent User Interfaces (IUI '19), March 17–20, 2019, Marina del Rey, CA, USA.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3301275.3302285

1 INTRODUCTION

This short paper investigates the use of *reinforcement learning* (RL) in *keystroke-level modelling* (KLM) [4] in human-computer interaction (HCI) research. Models such as Fitts' law, the power law of practice, KLM, GOMS, EPIC, SOAR, and ACT-R have been used to analyze interaction, estimate task performance, decrease errors and workload, and compare designs [3–5, 14, 15, 17, 24, 27]. KLM, in particular, was developed as a simplification of a cognitive model called GOMS. It omits cognitive functions like memory and attention and predicts *task completion time* as a linear function of elementary *operators*, such as pointing, homing, thinking, and waiting. This allows rapid evaluation of point-and-click type UIs. Despite, or thanks to, its simplicity, KLM is still popular (see e.g., [10]). It has been extended to model novel UIs like gestural input [9, 19].

We study RL as a method for learning a *task model*, or task policy, for a KLM. While task policies for UIs have been previously generated with shortest path algorithm [28], RL is not restricted to fully observable environments since the agent learns from interactions. A task policy specifies a sequence of actions (operators), or what the user is supposed to "do" with a UI: *"point the field, recall password, type password, point button, wait for system response, ..."*. KLM's task policies are often crafted by hand or by demonstration [10, 12]. However, given substantial variability in human behavior, this can be laborious. Moreover, because every design needs its own policy, this obstacle also limits the use of KLM in intelligent UIs and computational design.

RL is a general method for learning task policies in environments requiring sequential actions and where the environment is only partially known and not fully under the agent's control. It has attracted attention due to its generality and robustness, which have been demonstrated, among others, in gaming [22], robotics [18] and dialog systems [20]. An RL agent tries to discover optimal behavior by trial and error by interacting with the environment. During each iteration, the agent is given a reward, or penalty, which is negative reward. In essence, the reward function controls the learning of the agent. However, designing a reward function that gives raise to *human-like behavior* is a non-trivial problem [6, 16]. Motivations and other drivers of human behavior are difficult to learn from observations only [26]. The downside of data-driven approaches like inverse RL is that the obtained reward function is context-specific.

Two contributions are made towards the application of RL in KLMs: (1) we define the KLM as a *Markov Decision Process* (MDP), making it solvable with RL, and (2) demonstrate that plausible policies can be obtained in realistic interaction tasks. Here the key idea is to use KLM's operator costs (time costs) as negative reward when training the RL agent. To our knowledge, while RL has been studied in the context of full-fledged cognitive models [6, 16, 23], this is the first time it is used for KLMs. Besides automation of task policies, another benefit of RL is that the environment can be noisy, which enables simulating noisy sensors and motor outputs.

We report results from RL-KLM across three simple but representative interface problems using a plain vanilla RL method called Q-learning. RL-KLM could obtain a policy that represents an upper bound for performance possible with KLM. This may not be surprising, given the known link between RL and optimal control. However, this proof-of-concept is valuable. The learned

IUI '19, March 17–20, 2019, Marina del Rey, CA, USA

ACM ISBN 978-1-4503-6272-6/19/03.
https://doi.org/10.1145/3301275.3302285

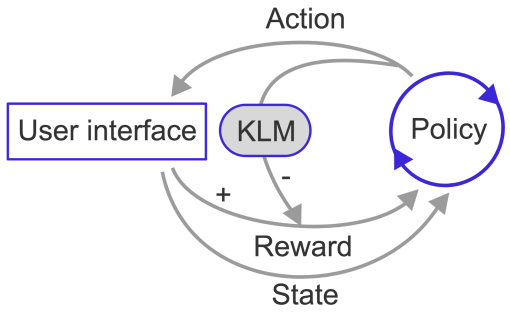

Figure 1: RL is here used to obtain task policies for KLMs. Task policies can be learned via RL when KLM is modeled as an MDP with UI-specified states and actions. We use the time costs of KLM operators as negative action rewards.

policies reflect how an experienced user might interact after extensive practice. The focus of this paper is automate the evaluation of the user interfaces, however, in the future RL-KLM may be used to as an objective function in intelligent UIs and computational design methods. To assess this possibility more concretely, we optimized an FSM (finite state machine) -based UI (remote controller). We conclude by discussing future work, especially the need to scale up the method and to assess its empirical validity.

2 RL-KLM

KLM is here defined as a Markov Decision Process (MDP). Time costs of the KLM operators define negative rewards for actions (Figure 1). A benefit of this approach is that only a state-action simulator and the KLM operators mapped to it are needed to represent the UI.

The resulting MDP can be solved using regular RL methods such as Q-learning. The obtained policy determines the action sequence that can then be deployed to assess the UI for task completion time. In addition to ability to simulate noise in input/ouput, extensibility is a potential benefit. Any memoryless operator type, even ones not yet specified in KLM literature, can be added to the MDP, assuming they can be unambiguously mapped to corresponding responses of the UI.

2.1 KLM as an MDP

The Markov Decision Process (MDP) is a memoryless process which is used to model sequential decision making [25]. An MDP is defined by the tuple $(S, A, P(a, s, s'), R(a, s, s'), \gamma)$, where S is a finite set of states, A is a finite set of actions and $P(a, s, s')$ defines the transition probabilities between the states. At each time step, the agent is in some state $s \in S$. The state can be changed to $s' \in S$ by an action $a \in A$. After each action, the agent receives a reward $r = R(a, s, s')$. The policy $\pi(s)$ defines which action is performed in each state. The agent's problem is to choose a policy $\pi(s)$, which maximizes cumulative rewards where the balance between immediate and future rewards is controlled by discount factor γ.

KLM is a linear model for estimating task completion time [4]. In the standard description, the user is not modeled as an agent making choices, but rather executing a prescribed sequence of actions (operators). Task completion time is the sum of time spent in actions (KLM operators) $t(a)$ that the agent must perform when interacting with the UI g_{UI} to solve task goal $g_{task} \in G_{tasks}$:

$$t(g_{task}, g_{UI}) = \sum_{i=1}^{I} t(a_i), \quad (1)$$

where I is a number of interactions. The original KLM [4] defined six operators, but many others have been added since. They share the property of being memoryless: the time cost of an operator is not dependent on anything else than the state of the UI.

When KLM is represented as an MDP, the user is modeled as an agent: At any time the agent is in a state defined by the UI, and has some actions a available, which are mapped to KLM operators O[1]. Actions change the state of the UI. The agent's goal is to change the UI to a specific state or visit certain states. The policy $\pi(s)$ tells which operators the agent should perform in which state to get to this goal.

To learn the policy via RL, from each action the agent receives a time penalty r defined by the reward function. Positive r can be attributed to successfully reaching the end-states, while KLM's operator durations define negative r (time costs). The state transitions $P(a, s, s')$ define the probability of transition between states. We use this to model the how likely it is that action is successful and the state changes match the user's expectations. If the probability is less than one, input or output errors may occur. This formulation requires no additions to the standard MDP. Moreover, a benefit of the MDP formulation is that it allows not only expressing cases with errors (e.g., speech recognition error) but any case where input to the system is not fully under user's control. However, learning a policy assuming noisy sensors will require on average more iterations to converge.

2.2 Solving the MDP with Reinforcement Learning

The optimal policy $\pi(s)$ can be obtained with a variety of RL methods [26], which generally work well when state-action spaces are not large. In this paper, we use the well-known ϵ-greedy Q-learning with episodic tasks for each task. In Q-learning, *expected* cumulative reward guides policy learning. It is defined for each state-action pair $Q(s, a)$. During the training, in each interaction step i, the learning agent selects an action a_i, moves to the state s_i and is rewarded with r_i. The Q-value is updated at each step by value iteration:

$$Q(s_i, a_i) \leftarrow Q(s_i, a_i) + \alpha \cdot (r_i + \gamma \cdot \max_a Q(s_{i+1}, a) - Q(s_i, a_i)), \quad (2)$$

where α is a learning rate and $\max_a Q(s_{i+1}, a)$ an estimate of the optimal future Q-value. The optimal policy finds a path to the goal from any starting state (assuming all states are reachable). It addresses how a user recovers from input/output errors that lead to unexpected (wrong) state changes.

2.3 Estimating Task Completion Time

Finally, task completion time for the given task and UI can be estimated by executing the learned policy. The policy, when exploration

[1]The wait operator is mapped to system state change.

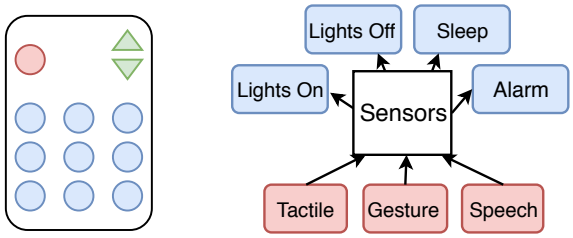

Figure 2: In Case 1 (left), RL-KLM must learn how to operate a remote controller using number buttons or arrow buttons. In Case 2 (right), it is learning to control a smart alarm clock with different (noisy) sensing modalities available.

(epsilon) is turned off, is a deterministic action sequence for the KLM model. It can be executed starting from any state of the UI.

To evaluate the whole UI g_{UI}, average task completion can be computed as a weighted sum using prior probabilities of all user tasks p_{task}:

$$t(G_{tasks}, g_{UI}) = \sum_{task=1}^{G_{tasks}} p_{task} t(g_{task}, g_{UI}). \quad (3)$$

3 EXPERIMENTS

We report results from three diverse cases. Our goal is not to evaluate the interfaces but to assess if RL-KLM can learn meaningful task policies. In Case 1, we look at a remote controller where the challenge is that it can be operated in redundant ways. In Case 2, we look at multimodal interaction with a smart alarm clock. The challenge is that it offers multiple alternative input devices, each with different sensing errors. In Case 3, we look at a form-filling UI. The challenge is that a more complex spatial trajectory must be learned to traverse the UI. An overview of the MDPs is provided in Table 1, and operator parameters are described below. Policies for each case was learned in 20 Q-learning iterations. RL-KLM code used for the experiments is available on GitHub[2].

3.1 KLM Parameters

We obtained KLM's tactile and gesture operator values from literature. Speech command durations are estimated with synthesized speech commands. Case 1 involves a user pressing physical buttons of the remote controller. We obtained operator costs from a previous paper [11]. The red and blue buttons are modeled with keyboard operator (.39 s) and green buttons with hot key operator (0.16 s). In the optimization case (next section), the first keystroke costs 1.18 s, including taking the controller and pressing a button. If the same button is pressed repeatedly, the cost is 0.13 s, and if a different, 0.36 s.

In Case 2, the speech operators are estimated by synthesizing spoken commands with speech synthesis tool Speech Synthesis Manager and measuring the duration. The commands and their durations are listed in the Table 1. In Cases 2 and 3, we utilize Fitts' law for the pointing operator [1], the tapping operator [8], and for the gesture operator [2]. Fitts' Law [21] estimates the user's

[2]https://github.com/aalto-speech/rl-klm

	Case 1	Case 2	Case 3
$\|S\|$	10	4	9
$\|A\|$	12	4 tactile, 4 gesture, 4 speech	9
O	Keystroke	Point, Gesture, Speech	Tap
$r(a,s,s')$	Literature	Fitts' Law, Synthesis	Fitts' Law
$p(a,s,s')$	1	0.2 ... 1	1

Table 1: Overview of Cases 1–3 as MDP tasks. O denotes KLM operators.

Movement Time (MT) between any two objects according to

$$MT = a + b \cdot \log_2\left(\frac{2D}{W}\right), \quad (4)$$

where a and b are case-dependent constants which are empirically estimated, D is the distance of the movement, and W is the width of the target object.

3.2 Case 1: Remote Controller

RL-KLM is here trained to control a television with a remote controller (Figure 2). There are ten states in the television: nine channels and the switched-off state. Channels are changed directly with blue (number) buttons. With green arrow buttons it is possible to move to next or previous channels. The red button turns the television off. We defined 90 tasks for reaching to all states from all states. The agent may use any of the 12 buttons.

Results: RL-KLM learned the fastest (possible) policy, which is to utilize both button types. When the target state can be reached with less than three presses of the green arrow buttons, the agent uses those, otherwise it uses the blue buttons. With this policy, average task completion time is 0.38 seconds. We conclude that RL-KLM learns the performance-optimal policy, which is to switch between the two button types when needed. By contrast, if only the green arrow buttons are allowed, best achievable task completion time is 0.43 seconds. If only the blue buttons are allowed, it is 0.42 seconds. The arrows are faster to execute (by 0.16 s per press), but all the states between the initial state and the goal must be visited.

3.3 Case 2: Multimodal Smart Alarm

This case looks at the effect of input/output recognition errors. The agent can turn the light on and off on the device, turn on the sleeping mode or turn the alarm on. Each command can be given using any of three modalities: tactile, gesture, and speech. Operator estimates are given in Table 2. We vary recognition error rates of gesture and speech sensors to assess if learned policies are sensitive to noise. A sensor may make two kinds of errors: 1) it may not detect input or 2) it may confuse it with another command within the modality.

Results: When no errors were present, not surprisingly, gestures and speech were fastest to use. KLM does not take into account the learning cost of commands. However, because the relative cost of correcting an error is large, this result changes when error rate

Figure 3: Case 2: When recognition error rates increase in speech and gesture sensors, the RL-KLM agent utilizes them less and eventually switches to use the tactile modality only.

increases. Figure 3 shows how the agent starts using less gestures and speech when recognition error rate increases. With gestures, task completion time was 1.6 s with zero error rate. When the recognition error rates increase (Figure 3), task completion time increases gradually from 1.8, 2.8, and 3.3 to 5.3 s. We conclude that RL-KLM can adapt its learned policies meaningfully in this case.

3.4 Case 3: Form-filling

In the form filling task, the task is not only to reach a certain state but to visit *all* nine states at least once and end by pressing 'Confirm'. The policies in this case are much longer and structured than in the others. The form UI is shown in Figure 4. The agent is able to visit any unvisited state at any given time. The policy agent learns defines the path between the form items. In the KLM, we do not include the cost of entering text as an operator, as this is inconsequential to policies.

Results: The agent learned to fill in the form without revisits and by using up and down movements as much as possible. The trajectory shows no loops and all items are covered. The trajectory is optimal, because vertical distances between items are shorter than horizontal distances. The learned policy is shown in Figure 4 with red arrows. We conclude that also a more complex spatial policy can be learned.

4 DESIGN OPTIMIZATION WITH RL-KLM

We used RL-KLM to optimize a simple remote controller with four states. We use RL-KLM as an objective and include two additional

	Tactile	Gesture	Speech
Lights on	0.984s	0.636s	0.817s
Lights off	1.184s	0.636s	0.828s
Alarm	3.984s	0.636s	0.849s
Sleep mode	3.984s	0.636s	1.067s

Table 2: Case 2: KLM operator estimates used in this case.

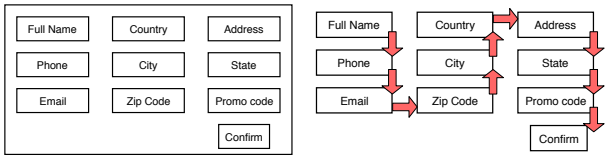

Figure 4: Case 3: When asked to learn how to fill in a form (left), RL-KLM learned the policy indicated by the red arrows.

objectives to regularize against unlearnably complex, performance-maximing policies. The objective is:

$$\min_{g_{UI} \in G_{UI}} \omega_t t(g_{UI}, G_{tasks}) + \omega_H H(g_{UI}) + \omega_C \sum^{G_{tasks}} C_{task}, \quad (5)$$

This objective function depends on the task completion time $t(g_{UI}, G_{tasks})$, the simplicity H of UI, and the consistency C of the task policies. The three terms are calibrated with ω_t, ω_H and ω_C. Hick-Hyman law [13, 27] defines the entropy of equally likely decisions n is defined as

$$H = \sum_{i}^{S} \log_2(n_i + 1). \quad (6)$$

In our case, the entropy is computed over all states S in the UI. The consistency measure C ensures that in the user interface the same operators are used in similar manner over all states:

$$C = -\sum_{i=1}^{|S|} \sum_{j=1}^{|A|} \log \left(\frac{c(a_j \rightarrow s_i)}{\sum_{k=1}^{|A|} c(a_k \rightarrow s_i)} \right), \quad (7)$$

where $|S|$ and $|A|$ are the number of states and actions, respectively, and $c(a_j \rightarrow s_i)$ is the number of times action a_j is used to transfer the state to s_i. Negative sign makes C positive as the result is always less or equal to zero. In this remote controller optimization case, $C = 0$ if the same buttons are always used to reach certain states, e.g., each state has dedicated button. If pressing a button results different state depending on the current state, $C > 0$.

4.1 Design Space and Optimization Approach

We use a Finite State Machine (FSM) to model a user interface mathematically for combinatorial optimization [7]. An FSM is defined by a state space S, an initial state s_0, and a transition matrix which defines the transitions between states. A single design is defined with a binary transition matrix and an action matrix. The action matrix defines which actions, in this case buttons, can be used for state transitions. To create a design space, we generate all transition matrices and for each matrix all possible action matrices. The generated design space consists designs with the different amount of buttons and transition designs. Most designs are either illogical and unusable. The problem with this approach is that the space grows exponentially as the number of states and actions increase. To reduce the size of the design space, we filter out infeasible designs (e.g., not every state is accessible from all states). In this work, because we are interested in quality of best-effort results, we use exhaustive search as the optimization method.

4.2 Results: Optimizing a Remote Controller

We optimized a remote controller with four states using the approach given above. When we optimized for simplicity ($\omega_t, \omega_C, \omega_H = 1$), the optimal user interface has only a single button which travels between states in sequence. Putting more weight on task completion time ($\omega_t = 5$), the optimal design had four buttons for each state. The consistency objective ensured that each state have their own unique button instead of buttons changing in each state.

5 CONCLUSION AND FUTURE WORK

Reinforcement learning offers a promising solution to the problem of task policies in KLM. We conclude that RL-KLM can learn plausibly human-like policies, although without further constraints they may remain too optimistic, and it remains open how well they represent real policies. However, this opens new vistas for, on the one hand, applying modern reinforcement learning methods in HCI and, applications of KLM in intelligent UIs and computational design, on the other. To facilitate future work in this area, our code is published on GitHub as a Python library together with the experiments. We see many possibilities to improve RL-KLM further. In this paper we used Q-learning, and were limited to rather small problem sizes. With novel RL algorithms it would be possible to increase the complexity of the user interfaces. We explored the use of RL-KLM in design optimization. The biggest issue with FSM optimization is to find feasible designs in exponentially increasing design space. Besides computational scalability, it remains a challenge for future work to extend the approach to mental operators, visual search requirements and so on, and to test the policies against empirical data.

ACKNOWLEDGEMENTS

KL's work was supported by Nokia Bell Labs, Academy of Finland project Co-adaptation, and Suomen Kulttuurirahasto. AO's work has been funded by the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No 637991).

REFERENCES

[1] Myroslav Bachynskyi, Gregorio Palmas, Antti Oulasvirta, Jürgen Steimle, and Tino Weinkauf. 2015. Performance and ergonomics of touch surfaces: A comparative study using biomechanical simulation. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems*. ACM, 1817–1826.

[2] Myroslav Bachynskyi, Gregorio Palmas, Antti Oulasvirta, and Tino Weinkauf. 2015. Informing the design of novel input methods with muscle coactivation clustering. *ACM Transactions on Computer-Human Interaction (TOCHI)* 21, 6 (2015), 30.

[3] Michael D Byrne. 2001. ACT-R/PM and menu selection: Applying a cognitive architecture to HCI. *International Journal of Human-Computer Studies* 55, 1 (2001), 41–84.

[4] Stuart K Card, Thomas P Moran, and Allen Newell. 1980. The keystroke-level model for user performance time with interactive systems. *Commun. ACM* 23, 7 (1980), 396–410.

[5] John M Carroll. 2003. *HCI models, theories, and frameworks: Toward a multidisciplinary science*. Elsevier.

[6] Xiuli Chen, Gilles Bailly, Duncan P Brumby, Antti Oulasvirta, and Andrew Howes. 2015. The emergence of interactive behavior: A model of rational menu search. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems*. ACM, 4217–4226.

[7] Kwang-Ting Cheng and Avinash S Krishnakumar. 1993. Automatic functional test generation using the extended finite state machine model. In *Design Automation, 1993. 30th Conference on*. IEEE, 86–91.

[8] Karim El Batran and Mark D Dunlop. 2014. Enhancing KLM (keystroke-level model) to fit touch screen mobile devices. In *Proceedings of the 16th international conference on Human-computer interaction with mobile devices & services*. ACM, 283–286.

[9] Orlando Erazo and José A Pino. 2018. Predicting user performance time for hand gesture interfaces. *International Journal of Industrial Ergonomics* 65 (2018), 122–138.

[10] Alix Goguey, Géry Casiez, Andy Cockburn, and Carl Gutwin. 2018. Storyboard-Based Empirical Modeling of Touch Interface Performance. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*. ACM, 445.

[11] Paul Holleis, Friederike Otto, Heinrich Hussmann, and Albrecht Schmidt. 2007. Keystroke-level model for advanced mobile phone interaction. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM, 1505–1514.

[12] Scott E Hudson, Bonnie E John, Keith Knudsen, and Michael D Byrne. 1999. A tool for creating predictive performance models from user interface demonstrations. In *Proceedings of the 12th annual ACM symposium on User interface software and technology*. ACM, 93–102.

[13] Ray Hyman. 1953. Stimulus information as a determinant of reaction time. *Journal of experimental psychology* 45, 3 (1953), 188.

[14] Melody Y Ivory and Marti A Hearst. 2001. The state of the art in automating usability evaluation of user interfaces. *ACM Computing Surveys (CSUR)* 33, 4 (2001), 470–516.

[15] Bonnie E John and David E Kieras. 1996. The GOMS family of user interface analysis techniques: Comparison and contrast. *ACM Transactions on Computer-Human Interaction (TOCHI)* 3, 4 (1996), 320–351.

[16] Jussi PP Jokinen, Sayan Sarcar, Antti Oulasvirta, Chaklam Silpasuwanchai, Zhenxin Wang, and Xiangshi Ren. 2017. Modelling Learning of New Keyboard Layouts. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*. ACM, 4203–4215.

[17] Davis E Kieras and Davis E Meyer. 1997. An overview of the EPIC architecture for cognition and performance with application to human-computer interaction. *Human–Computer Interaction* 12, 4 (1997), 391–438.

[18] Jens Kober, J Andrew Bagnell, and Jan Peters. 2013. Reinforcement learning in robotics: A survey. *The International Journal of Robotics Research* 32, 11 (2013), 1238–1274.

[19] Ahreum Lee, Kiburm Song, Hokyoung Blake Ryu, Jieun Kim, and Gyuhyun Kwon. 2015. Fingerstroke time estimates for touchscreen-based mobile gaming interaction. *Human movement science* 44 (2015), 211–224.

[20] Esther Levin, Roberto Pieraccini, and Wieland Eckert. 2000. A stochastic model of human-machine interaction for learning dialog strategies. *IEEE Transactions on speech and audio processing* 8, 1 (2000), 11–23.

[21] I Scott MacKenzie. 1992. Fitts' law as a research and design tool in human-computer interaction. *Human-computer interaction* 7, 1 (1992), 91–139.

[22] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. 2013. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602* (2013).

[23] Shelley Nason and John E Laird. 2005. Soar-RL: Integrating reinforcement learning with Soar. *Cognitive Systems Research* 6, 1 (2005), 51–59.

[24] Fabio Paterno. 2012. *Model-based design and evaluation of interactive applications*. Springer Science & Business Media.

[25] Martin L Puterman. 2014. *Markov decision processes: discrete stochastic dynamic programming*. John Wiley & Sons.

[26] Stuart Russell. 1998. Learning agents for uncertain environments. In *Proceedings of the eleventh annual conference on Computational learning theory*. ACM, 101–103.

[27] Steven C Seow. 2005. Information theoretic models of HCI: a comparison of the Hick-Hyman law and Fitts' law. *Human-Computer Interaction* 20, 3 (2005), 315–352.

[28] Harold Thimbleby, Paul Cairns, and Matt Jones. 2001. Usability analysis with Markov models. *ACM Transactions on Computer-Human Interaction (TOCHI)* 8, 2 (2001), 99–132.